

# 前 言

这是一本介绍 MATLAB 仿真应用的参考书，适合于使用 MATLAB 或正在打算使用 MATLAB 的读者，尤其是那些想使用 Simulink 对动态系统建模的读者。

MATLAB 是一种面向科学与工程计算的高级语言，它集科学计算、自动控制、信号处理、神经网络和图像处理等学科的处理功能于一体，具有极高的编程效率。MATLAB 是一个高度集成的系统，MATLAB 提供的 Simulink 是一个用来对动态系统进行建模、仿真和分析的软件包，它支持线性和非线性系统，能够在连续时间域、离散时间域或者两者的混合时间域里进行建模，它同样支持具有多种采样速率的系统。在过去几年里，Simulink 已经成为教学和工业应用中对动态系统进行建模时使用得最为广泛的软件包。

MATLAB 的最新版本是 MATLAB6.0，对应的 Simulink 版本是 4.0，这也是本书编写所依据的版本。由于 MATLAB6.0 对用户的计算机配置要求比较高，所以有些用户仍然在使用 MATLAB5.3。本书同时也兼顾了这部分读者的需求，书中的大部分内容，在两种版本下都是适用的，对于两种版本区别较大的地方，本书都分别进行了说明。

全书共分为 10 章，各章的基本内容为：

第一章简要介绍了计算机仿真的基本概念和基本方法。第二章概括地介绍了 MATLAB 语言的基本特点和编程技巧，并以大量的例子向读者展示如何用 MATLAB 实现静态仿真。

第三章至第八章则全面细致地介绍了 Simulink 的使用。其中，第三章提供了使用 Simulink 建模的入门知识。第四章详细讲解了 Simulink 的建模技巧，包括建立子系统、封装子系统和建立条件执行模块等。第五章介绍了一些 Simulink 比较深入的主题，通过本章，读者可以了解 Simulink 工作原理及其他知识。第六章介绍如何为仿真模型设置参数和分析仿真的结果。Simulink 和其他高级语言一样也提供了调试工具，它的使用方法是第七章的主题。第八章给出了部分模块的使用参考，读者可以获得这些模块的详尽信息。

第九章系统全面讲解了如何使用 S-函数对 Simulink 进行扩展。它将教会读者如何用 M 语言、C 语言和 C++ 编写 S-函数。

第十章简要讲解了如何使用 Real-Time Workshop 软件包。经过这一章的学习，读者可以学到如何从 Simulink 模型中生成可单机运行的实时代码，如何建立外部模式等知识。

最后给出了一个 MATLAB 常用函数的附录，它依据 MATLAB6.0 整理而成。

本书的编写得到了汪小平的大力支持，在此表示衷心的感谢。

由于时间紧迫，加上作者水平有限，书中难免有遗漏、错误与不当之处，敬请读者批评指正。

作者

2001 年 3 月

# 目 录

第一章 计算机仿真概论 .....	1
1.1 计算机仿真基本概念 .....	1
1.1.1 什么是计算机仿真 .....	1
1.1.2 计算机仿真模型与方法 .....	2
1.1.3 计算机仿真的步骤 .....	3
1.2 一个实例——报童问题仿真 .....	5
1.3 随机变量的产生 .....	8
1.3.1 均匀分布随机数的产生 .....	8
1.3.2 随机变量的产生 .....	8
1.4 输入数据的分析 .....	11
1.5 离散系统仿真 .....	14
1.5.1 离散系统概述 .....	14
1.5.2 离散系统仿真的基本方法 .....	16
第二章 用 MATLAB 实现静态仿真 .....	19
2.1 MATLAB 基础 .....	19
2.1.1 为什么选用 MATLAB .....	19
2.1.2 MATLAB 基本特性 .....	22
2.1.3 MATLAB 的三种执行方式 .....	24
2.1.4 MATLAB 里的函数 .....	28
2.1.5 MATLAB 里的矩阵（数组）运算 .....	30
2.1.6 MATLAB 里的程序设计 .....	38
2.1.7 使用 MATLAB 在线帮助 .....	40
2.2 仿真应用：输入数据分析 .....	43
2.2.1 随机变量的产生 .....	43
2.2.2 输入数据的分析 .....	47
2.3 仿真应用之输出分析 .....	54
2.3.1 图形函数 .....	54
2.3.2 曲线拟合与插值 .....	59
2.4 仿真应用实例 .....	63
2.4.1 二进制通信系统的蒙特卡罗仿真 .....	64
2.4.2 排队系统仿真 .....	67

<b>第三章 Simulink 入门</b>	75
3.1 Simulink 简介	75
3.1.1 什么是 Simulink	75
3.1.2 Simulink 模型特点	77
3.2 创建一个简单的模型	80
3.3 熟悉 Simulink 模型窗口	83
3.4 键盘和鼠标操作总览	88
3.5 模块库简介	90
<b>第四章 Simulink 详解</b>	95
4.1 Simulink 的模块和模块库	95
4.1.1 Simulink 里的模块	95
4.1.2 Simulink 的模块库	99
4.2 模拟方程	103
4.3 Simulink 里的数据类型	106
4.3.1 Simulink 支持的数据类型	106
4.3.2 数据类型传播	108
4.3.3 在模型里使用复数信号	109
4.4 建立子系统	110
4.4.1 建立子系统	110
4.4.2 用子系统来自定义库	112
4.5 封装子系统	114
4.5.1 子系统封装示例	114
4.5.2 initialization 页	117
4.5.3 icon 页（图标页）	122
4.5.4 documentation 页	126
4.5.5 为封装的模块建立动态对话框	127
4.6 建立条件子系统	129
4.6.1 使能子系统	129
4.6.2 触发子系统	131
4.6.3 触发使能子系统	132
<b>第五章 深入理解 Simulink</b>	135
5.1 Simulink 如何工作	135
5.1.1 基本模型	135
5.1.2 进行仿真	136
5.1.3 过零检测	138

5.2	代数环 .....	141
5.2.1	直接馈入环路 (direct feedthrough)——代数环 .....	141
5.2.2	非代数直接馈入环路 .....	143
5.3	离散时间系统 .....	144
5.4	使用回调函数 .....	148
5.4.1	回调函数基本概念 .....	148
5.4.2	回调函数示例 .....	152
5.5	模型文件格式 .....	154
<b>第六章</b>	<b>仿真运行和结果分析 .....</b>	<b>167</b>
6.1	使用菜单命令运行仿真 .....	167
6.2	仿真参数对话框 .....	168
6.2.1	Solver 页 .....	169
6.2.2	Workspace I/O 页 .....	174
6.2.3	Diagnostics 页 .....	180
6.2.4	Advanced 页 .....	182
6.3	改善仿真的性能和精确度 .....	183
6.3.1	加速仿真 .....	183
6.3.2	改善仿真的精度 .....	184
6.4	从命令行运行仿真 .....	184
6.4.1	使用 sim 命令 .....	184
6.4.2	使用 set_param 命令 .....	185
6.5	分析仿真结果 .....	186
6.5.1	观看输出结果的轨迹 .....	186
6.5.2	线性化 .....	188
6.5.3	平衡点的分析 .....	191
<b>第七章</b>	<b>Simulink 调试器 .....</b>	<b>197</b>
7.1	使用调试器 .....	197
7.2	增量运行模型 .....	200
7.3	设置断点 .....	202
7.3.1	非条件中断 .....	202
7.3.2	条件中断 .....	204
7.4	显示仿真有关的信息 .....	205
7.4.1	显示模块的输入输出 I/O .....	206
7.4.2	显示代数环信息 .....	209
7.4.3	显示系统状态 .....	209
7.4.4	显示积分信息 .....	210
7.5	显示模型的信息 .....	210



7.6	Simulink4.0 的图形调试工具 .....	213
7.7	调试命令使用参考 .....	215
<b>第八章</b>	<b>模块使用参考 .....</b>	<b>223</b>
<b>第九章</b>	<b>用 S-函数扩展 Simulink .....</b>	<b>257</b>
9.1	S-函数综述 .....	257
9.1.1	什么是 S-函数 .....	257
9.1.2	S-函数如何工作 .....	260
9.1.3	M 文件和 C MEX 文件 S-函数综述 .....	262
9.1.4	S-函数概念 .....	263
9.2	建立 M 文件 S-函数 .....	265
9.2.1	如何使用模板 .....	265
9.2.2	定义 S-函数的初始信息 .....	268
9.2.3	输入和输出参量说明 .....	270
9.2.4	M 文件 S-函数的几个示例 .....	271
9.3	C MEX S-函数简介 .....	282
9.3.1	介绍 .....	282
9.3.2	编写基本的 C MEX S-函数 .....	283
9.3.3	建立更复杂的 C MEX S-函数 .....	289
9.4	建立 C++ S-函数 .....	291
9.4.1	源文件格式 .....	292
9.4.2	建立永久 C++ 对象 .....	295
<b>第十章</b>	<b>使用 Real-Time Workshop .....</b>	<b>297</b>
10.1	Real-Time Workshop 综述 .....	297
10.1.1	Real-Time Workshop 能做什么 .....	297
10.1.2	使用前的准备工作 .....	298
10.1.3	RTW 中的基本概念 .....	299
10.2	生成普通的实时程序 .....	301
10.2.1	Simulink 模型 .....	301
10.2.2	生成实时代码 .....	302
10.2.3	代码验证 .....	307
10.3	代码生成和建立过程 .....	310
10.3.1	自动程序建立 .....	310
10.3.2	Real-Time Workshop 用户界面 .....	311
10.4	外部模式 .....	316
10.4.1	介绍 .....	316
10.4.2	使用 grt (普通实时目标) 的外部模式入门 .....	317

10.4.3	外部模式 GUI .....	320
10.4.4	外部模式的 TCP/IP 实现 .....	324
10.5	RTW 代码库 .....	325
10.5.1	Custom Code Library (自定义代码库) .....	325
10.5.2	使用自定义代码模块示例 .....	327
附录	<b>MATLAB</b> 函数参考 .....	331
参考文献	.....	345

# 第一章 计算机仿真概论

## 1.1 计算机仿真基本概念

### 1.1.1 什么是计算机仿真

#### 1. 仿真的定义

仿真的基本思想是利用物理的或数学的模型来类比模仿现实过程，以寻求过程和规律。它的基础是相似现象，相似性一般表现为两类：几何相似性和数学相似性。当两个系统的数学方程相似，只是符号变换或物理含义不同时，这两个系统被称为“数学同构”。事实上，相似性是一个含义比较广的概念，既有几何形状的相似，结构的相似，功能的相似，还有机理和联想的相似，后者尤其是创造性的源泉。

仿真的方法可以分为三类：

(1) 实物仿真。它是对实际行为和过程进行仿真，早期的仿真大多属于这一类。物理仿真的优点是直观、形象，至今在航天、建筑、船舶和汽车等许多工业系统的实验研究中心仍然可以见到。比如：用沙盘仿真作战，利用风洞对导弹或飞机的模型进行空气动力学实验、用图纸和模型模拟建筑群等都是物理仿真。但是要为系统构造一套物理模型，不是一件简单的事，尤其是十分复杂的系统，将耗费很大的投资，周期也很长。此外，在物理模型上做实验，很难改变系统参数，改变系统结构也比较困难。至于复杂的社会、经济系统和生态系统就更无法用实物来做实验了。

(2) 数学仿真。就是用数学的语言、方法去近似地刻画实际问题，这种刻画的数学表述就是一个数学模型。从某种意义上，欧几里德几何、牛顿运动定律和微积分都是对客观世界的数学仿真。数学仿真把研究对象（系统）的主要特征或输入、输出关系抽象成一种数学表达式来进行研究。数学模型可分为：

- 解析模型（用公式、方程反映系统过程）；
- 统计模型（蒙特卡罗方法）；
- 表上作业演练模型。

然而数学仿真也面临一些问题，主要表现在以下几个方面：

- 现实问题可能无法用数学模型来表达，即刻画实际问题的表达式不存在或找不到；
- 找到的数学模型由于太复杂而无法求解；
- 求出的解不正确，可能是由模型的不正确或过多的简化近似导致的。

(3) 混合仿真。又称为数学-物理仿真，或半实物仿真，就是把物理模型和数学模型以及实物联合在一起进行实验的一种方法，这样往往可以获得比较好的结果。

## 2. 计算机仿真

计算机仿真也称为计算机模拟，就是利用计算机对所研究系统的结构、功能和行为以及参与系统控制的主动者——人的思维过程和行为，进行动态性的比较和模仿，利用建立的仿真模型对系统进行研究和分析，并可系统过程演示出来。

自 20 世纪 40 年代以来，用计算机方法去研究系统的特性成为科学发展的时尚。在计算机上对构成的系统模型进行试验，为模型的建立和实验提供了巨大灵活性和方便性。利用计算机，使得数学模型的求解变得更加方便、快捷和精确，能解决的问题的领域也大大扩展了。计算机仿真特别适合于解决那些规模大，难以解析化以及不确定的系统。

### 1.1.2 计算机仿真模型与方法

#### 1. 系统

系统是指相互联系又相互作用的元素之间的有机组合。这里所指的系统是广义的，它包含所有的工程系统和非工程系统。电气、机械和通信系统都是工程系统，而经济、交通、管理和生物系统等都是非工程系统。

任何系统都存在三方面需要研究的内容：即实体、属性和活动。

实体：组成系统的具体对象。

属性：实体的特性（状态和参数）。

活动：对象随时间推移而发生的状态变化。

对于工厂系统而言（图 1-1），系统的实体是工厂的部门、定单和产品；它的属性是部门类型、定单数量和各部门的设备数量；它的活动则是各个部门的计划、采购、装配和销售过程。

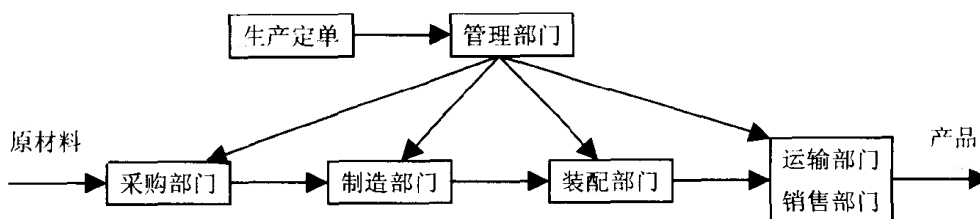


图 1-1 工厂系统组成

由于组成系统的实体之间相互作用而引起实体属性的变化，通常用“状态”的概念来描述。研究系统就是研究系统状态的改变，即系统的转变。

系统研究除了研究系统的实体、属性和活动之外，还应研究系统的环境。环境是指对系统的活动结果产生影响的外界因素。自然界的一切事物都存在着相互联系和相互影响，而系统是在外界因素不断变化的环境中演变发展的，因此，环境因素是必须考虑的。对开放的非工程系统更是如此。

系统与环境的边界往往不易确定，它们随研究的目的而异。例如：工厂系统的订货问题既可以将其视为环境对生产产生的影响，也可以将定货看作纳入系统内的活动。

系统研究包括系统分析、系统综合及系统预测等三个方面。

系统具有以下四个主要特性：

（1）目的性。即设计和运行某一系统是为了实现一定的目的，它包括两个相互紧密联系

的含义，即实现某些特定功能和系统优化。

(2) 集合性。系统的各个组成部分（元素或子系统）之间具有一定的独立性，但它们同时构成一个有机整体。

(3) 相关性。组成系统的子系统间相互联系，相互作用，某一子系统的输入则是与之相联系的前一子系统的输出。为使系统正常运行各子系统间存在一定的逻辑关系。

(4) 环境适应性。任何系统都有确定的边界和环境，系统从外部环境接受输入（包括正常输入和随机干扰），经过系统转换再向外部环境产生输出。由于外部环境是变化的，为了使系统优化，系统生存必须进行相应调节使之适应环境的变化。

## 2. 模型

模型是对现实系统有关结构信息和行为的某种形式的描述。这种描述是对那些有用的和令人感兴趣的特性的抽象化与简化。模型在所研究系统的某一侧面具有与系统相似的数学描述或物理描述。

模型建立的任务是要确定模型的结构和参数。建立模型有三种途径：

(1) 对内部结构和特性清楚的系统，即所谓的白箱（多数的工程系统都属于这一类），可以利用已知的一些基本规律，经过分析和演绎导出系统模型。

(2) 对那些系统结构和特性不清楚或不很清楚的系统，即所谓的黑箱或灰箱，如果允许直接进行试验性观测，则可以先猜想模型再通过试验验证和修正之。

(3) 对那些系统结构和特性不清楚或不很清楚但又不允许直接实验观测的系统（非工程系统多属于这一类），则采用数据收集和统计归纳的方法来假设模型。

在选择模型结构时，要以方便达到模型研究的目的为前提。通常遵循下述原则：

(1) 相似性。即模型与被研究系统具有相似的数学描述或物理特征。相似原则是选择模型最重要的原则。

(2) 简单性。一般而言，在实用的前提下，模型越简单越好。

(3) 切题性。模型应该针对研究目的的有关方面，而不是一切方面。

(4) 吻合性。模型结构的选择，应尽可能对利用的数据作合理的描述，通常其实验数据应尽可能由模型来解释。

(5) 综合精度。它是模型框架、结构和参数集合的一种指标。若有限的信息限制了模型的精度，则应进行各方面精度的平衡和折中。

(6) 可辨识性。模型结构必须选择可辨识的形式。若一个结构具有无法估计的参数，则此结构无实用价值。

### 1.1.3 计算机仿真的步骤

计算机仿真，概括地说是一个“建模—实验—分析”的过程，即仿真不单纯是对模型的实验，还包括从建模到实验再到分析的全过程。因此进行一次完整的计算机仿真有以下步骤：

#### (1) 列举并列项目

每一项研究都应从说明问题开始，问题由决策者提供，或由熟悉问题的分析者提供。

#### (2) 设置目标及完整的项目计划

目标表示仿真要回答的问题、系统方案的说明。项目计划包括人数、研究费用以及每一阶段工作所需时间。

### (3) 建立模型和收集数据

模型和实际系统没有必要一一对应，只需描述实际系统的本质。因此，最好从简单的模型开始，然后建立更复杂的模型。

### (4) 编制程序和验证

利用数学公式、逻辑公式和算法等来表示实际系统的内部状态和输入/输出的关系。建模者必须决定是采用通用语言如 FORTRAN、C 还是专用仿真语言来编制程序。在本书中，我们的选择是 MATLAB 和其动态仿真工具 Simulink，在以下的章节中，我们会逐步展示使用 MATLAB 的好处。

### (5) 确认

确认指确定模型是否精确地代表实际系统。它不是一次完成，而是比较模型和实际系统特性的差异，不断对模型进行校正的迭代过程。

### (6) 实验设计

确定仿真的方案、初始化周期的长度、仿真运行的长度以及每次运行的重复次数。

### (7) 生产性运行和分析

通常用于估计被仿真系统设计的性能量度。利用理论定性分析、经验定性分析或系统历史数据定量分析来检验模型的正确性，利用灵敏度分析等手段来检验模型的稳定性。

### (8) 文件清单和报表结果

### (9) 实现

图 1-2 是整个计算机仿真的程序图。

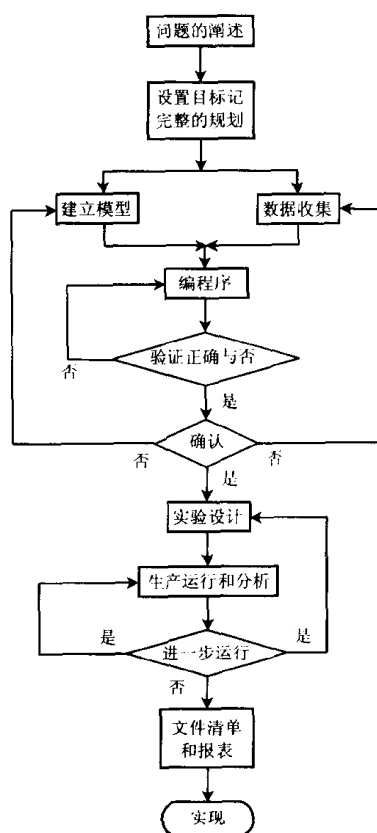


图 1-2 计算机仿真程序流程图

## 1.2 一个实例——报童问题仿真

报童问题是一个古典的概率统计分析问题，虽然问题本身并不复杂，但作为一个演示实例，可以反映出计算机仿真的很多特征。

### 1. 报童问题

一报童从报刊发行中心定报后零售，每卖一份报纸可赚钱  $a$  元，若定报后卖不出去，则可再退回发行处，此时每退一份报要赔钱  $b$  元。虽然每天卖出报的份数是随机的，但报童可根据以往卖报情况的统计来获得每天卖  $k$  份的概率  $p(k)$ ，试求报童每天期望受益达到最大的定报量  $z'$ 。

### 2. 数学模型

设报童每天订报  $z$  份，而报纸每天卖出  $y$  份，我们假设  $y$  的分布为

$$p(y=k) = p_k \quad k=0, 1, 2, \dots$$

考虑到报童每天的损失有如下两种情形。

(1) 供过于求。因退货造成的平均损失为：

$$c_1 = b \sum_{k=0}^{\infty} (z-k) p_k$$

(2) 供不应求。因缺货造成的平均损失为：

$$c_2 = a \sum_{k=z+1}^{\infty} (k-z) p_k$$

所以，每天的期望损失费（也可以从总收益的角度来考虑）为

$$C(z) = c_1 + c_2$$

现在我们的目标是求出使得每天期望损失最小的定报量，换言之，就是使报童的每天期望总收益达到最大。写成一个目标函数的形式

$$z' = C_{\min}^{-1}(z)$$

约束条件如  $z$  的取值范围，要受到报童的资本多少的影响。

只有在特殊的概率分布情况下，我们才可以推导出  $C(z)$  的解析形式，并通过求极值的方法来求解。但在实际的应用中，这样的思路往往是行不通的。可以通过枚举所有可能的订报量，求出对应的平均损失，进行比较求出满足条件的  $z$ ，这里搜索域通常是有限的。上面说的就是一个比较简单的计算机仿真方法。

### 3. 报童问题的计算机仿真

对于给定每一订报量  $Z$  值，利用离散随机变量采样算法产生给定分布的随机数  $R$ ，用来表示报童当天卖出的报纸数，从而可以计算出一天的损失以及一个阶段的平均损失。这里比较关键的一点是如何产生服从给定分布的随机变量，这个内容在本章的第二节会有详尽的介

绍。而且在实际的应用中，分布并非总是给定了的，需要我们收集数据，并从中辨识分布，进行参数估计，这部分的知识将在本章的第三节讨论。图 1-3 是根据上述思路设计出的仿真流程框图。

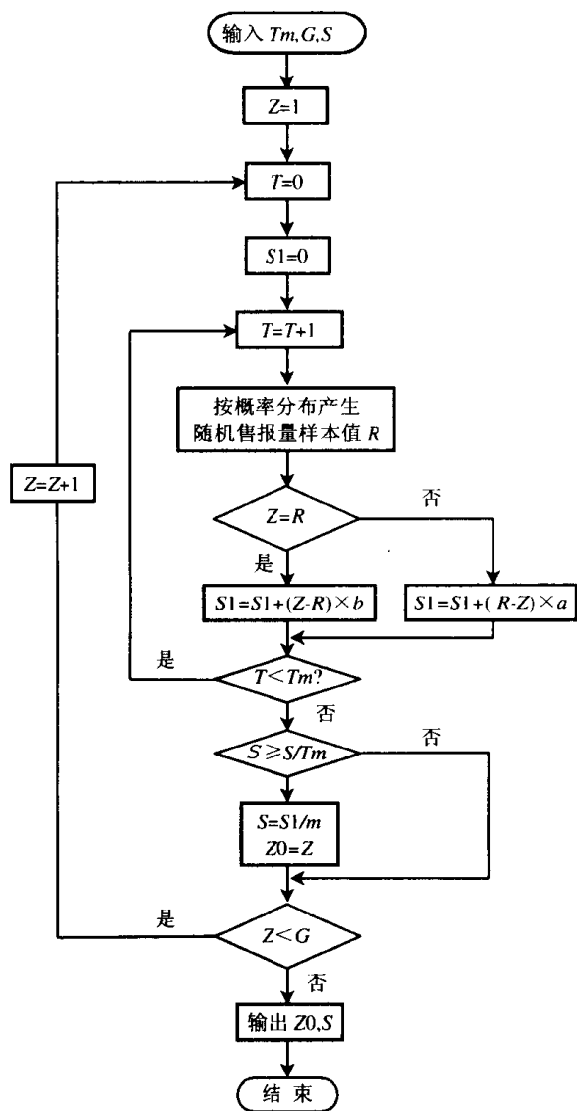


图 1-3 报童问题计算机仿真流程框图

其中各变量含义如下：

$Tm$ ——一轮试验的预定模拟天数

$T$ ——一轮实验的仿真天数累计值

$Z$ ——订报量

$Z'$ ——最优订报量

$G$ ——定报量  $Z$  之上界

$S1$ ——损失值之累计值

$S$ ——最小损失值

这里， $a$  和  $b$  是这个问题的两个参数。

#### 4. 计算机仿真程序

根据图 1-3 所示框图，我们不难写出报童问题的仿真程序，其 MATLAB 源码附在下面，供读者参考。如何用 MATLAB 来编写仿真程序，不是这一章的重点。



```

function [superz,supers]=baotong (tm,g,a,b)
z=1;
supers=1000;
while z<g
    % r=poissdis (5,1,tm) ;
    r = round (2*randn (1,tm) +5) ;
    % 产生均匀分布随机数

    t = 1;
    s = 0;
    dv = z>r;
    s = sum (((z-r)*b).*(dv));
    s=s+sum(((r-z)*a).*(1-dv));
    aver_s = s/tm;
    if supers >= aver_s;
        supers = aver_s;
        superz = z;
    end;
    z = z+1;
end;

```

上面的代码，是在均匀分布的假设下编写的，对于其他的分布，读者只要用相应的随机数发生器进行替换即可，本章的第二节将会介绍生成不同分布的随机数的方法，而在第二章则会讨论如何在 MATLAB 里产生随机数。

### 5. 仿真结果和输出数据分析

在分布为均匀分布，参数  $a=0.2$ ， $b=0.4$  的条件下，在 MATLAB 命令窗口运行仿真程序就可以得出仿真的结果。例如

```

>> [z,s]=baotong (5,10,0.2,0.4)
z =
    4
s =
    0.2400

```

此外，我们还可以改变参数  $a$ 、 $b$  的值，观察相应最优值的变化，用 MATLAB 可以很方便的画出  $C-a$ ， $C-b$  的变化曲线。

### 6. 报童问题模拟系统的推广和应用

报童仿真问题的改进推广和应用可以有以下几个方面：

(1) 求每天的卖出报数服从任意分布的情况下，使报童收益最大意义下的最优订报量  $Z'$ ；

- (2) 对报纸总发行量进行测算;
- (3) 适当修改仿真系统, 可将其用于企业的订货和库存策略研究。

## 1.3 随机变量的产生

计算机仿真能够成功的关键是在计算机上实现真正的随机采样, 而随机采样产生的基础是随机数。例如我们前面所举的报童问题的仿真以及本书后面将要提到的实例, 服从给定分布的随机数的产生都是很重要的问题。

这一节假定分布函数已经确定, 而主要介绍产生具有这种分布的随机变量的方法。这些方法的基础是独立均匀分布  $U(0,1)$  的随机源, 在 MATLAB 中提供了产生  $U(0,1)$  分布的随机数的函数。此外, 它还提供了产生各种分布随机数的函数, 如正态分布和泊松分布等等。这些函数的使用方法我们在第二章详细介绍, 这里我们把注意力放在它们的生成原理上。

### 1.3.1 均匀分布随机数的产生

随机数的产生较为复杂, 它的研究涉及抽象的代数和数论以及系统程序设计和计算机硬件工程等多种学科。

现在应用的大多数随机数发生器是各种同余发生器, 它由 Lehmer 在 1951 年提出, 根据下面的递推公式产生 0 到  $m-1$  之间的整数序列  $X_1, X_2, X_3, \dots$

$$X_{i+1} = (a X_i + c) \bmod m$$

在上式中,  $c$  不为 0 时, 这种方法就是混合同余法;  $c$  为 0 则是乘同余法。由于新近对混合同余法所期望的性能改善没有得到明确的证明, 所以今天所使用线性同余发生器都是性质被研究得比较透彻的乘同余法。对于二进制机器, 可以按以下规则选择  $a$  和  $m$ :

(1) 取  $m = 2^j$ ,  $j$  是某个整数, 一般  $m$  选择在机器所能表示的数的范围内。由于上式生成的伪随机序列的周期为  $m/4$ , 所以选择时应使这个周期值大于仿真实验的持续期。

(2)  $a$  一般取与  $a \approx 2^{p/2}$  最接近而又满足  $a = 8K + 3$  的整数, 其中  $K$  为任意整数,  $p$  为机器字长。

例如, 希望产生一个 8000 个数的序列 (最小单位为 1), 那么根据法则 (1),  $m$  应接近 32000, 如果机器字长为 16, 则可以选择  $m = 2^{15}$ , 再根据法则 (2),  $a \approx 2^8$ , 与此最接近的满足条件的数是 181, 即  $a$  取 181。

用上面方法产生的伪随机数基本上符合均匀分布的统计特性, 其均值为  $m/2$ , 方差为  $m^2/12$ 。为了得到 0 到 1 之间的随机数, 取  $X_i/m$  ( $i=0,1,\dots$ )。

### 1.3.2 随机变量的产生

几乎所有产生随机变量的方法都可以按照它们的理论基础推导出来。

#### 1. 反函数法

反函数是一种被广泛应用、产生服从给定分布的随机变量的方法。它的理论基础是概率积分变换原理, 其叙述如下: 设  $X$  是一个在 0 到 1 间均匀分布的随机变量  $U(0,1)$ , 其采样

值为  $x$ ，而  $Y = F^{-1}(X)$  为满足概率分布  $F(y)$  的随机变量。这个原理的证明是十分容易的，可以在各种概率书籍里找到。因此产生变量  $Y$  的随机数  $y$  可以分为两步：

(1) 产生  $U(0,1)$  的随机数  $x$ ；

(2)  $y = F^{-1}(x)$ 。

下面举几个例子，来说明这种方法，假定  $x$  为已产生  $U(0,1)$  的随机数序列。

(1) 服从分布  $U(a,b)$  随机数  $y$  的产生，即  $Y$  在区间  $[a,b]$  上均匀分布，它的分布函数  $F(y)$

$$F(y) = \begin{cases} 0 & y < a \\ (y-a)/(b-a) & a \leq y \leq b \\ 1 & y > b \end{cases}$$

产生公式

$$y = (b-a)x + a$$

(2) 指数分布

$$F(y) = \begin{cases} 1 - e^{-\lambda y} & , y \geq 0 \\ 0 & , y < 0 \end{cases}$$

不难推出其产生公式

$$y = -\frac{1}{\lambda} \ln(1-x)$$

上式中通常用  $x$  来替代  $1-x$ 。

(3) 韦伯分布

韦伯分布在可靠性问题中有广泛的应用，例如机械、电气零件的失效时间分布常用韦伯分布来表示，其分布函数为

$$F(y) = \begin{cases} 1 - \exp[-(\frac{y-v}{\alpha})^\beta] & , y \geq v \\ 0 & , y < v \end{cases}$$

相应得到

$$y = v + \alpha[-\ln(1-x)]^{1/\beta}$$

## 2. 卷积法

有一些重要的分布的随机变量  $Y$ ，可以表示为独立同分布的其他随机变量  $X_i$  之和，这样可以通过求  $X_i$  的采样值  $x_i$  之和来产生  $Y$  的采样值。之所以把这种方法称为卷积法，是从密度函数之间的关系来考虑的。典型的例子有：

(1)  $m$  爱尔兰分布

我们知道具有均值  $\beta$  的  $m$  爱尔兰随机变量，可以表示为  $m$  个独立同分布的指数随机变量  $X_i$  之和，指数分布的参数为  $\beta/m$ 。这样就可以先按照前面的方法产生  $m$  个符合指数分布的随机数，然后将它们求和即可。

(2) 三角形分布

在  $[-1,1]$  上按三角形分布的随机变量  $Z$  同样可以表示为两个独立的随机变量之和， $Z=X+Y$ ，

其中  $X, Y$  分别是  $[0,1]$  和  $[-1,0]$  上的均匀随机变量。据此不难产生  $Z$  的采样值。

### 3. 产生正态分布随机数

标准正态分布随机数可以用两种方法产生。

#### (1) 近似法

这种方法的基础是概率论中的中心极限定理。可以用多个互相独立的均匀分布随机数来产生正态分布，比较著名的一种是随机数的个数选 12 个，即

$$y = \sum_{i=0}^{12} x_i - 6$$

不难验证其均值和方差分别为 0 和 1。这种方法的缺点是精度不够，尽管可以通过提高被加和随机变量的个数来提高精度，却降低了效率。

#### (2) 直接变换法

因为正态分布的反函数不能用解析式表示出来，所以它不能用反函数来产生，下面介绍直接变换法比近似法更精确有效，这里我们只介绍其操作，不解释它的得来。该方法的步骤如下。

- 用两个独立的均匀分布随机数产生器，分别产生随机数  $x_1$  和  $x_2$ 。
- 再根据下面两式进行计算：

$$y_1 = \sqrt{(-2\ln x_1)} \cdot \cos(2\pi x_2)$$

$$y_2 = \sqrt{(-2\ln x_1)} \cdot \sin(2\pi x_2)$$

所得到的  $y_1$  和  $y_2$  是两个独立的标准正态随机数，取其中之一即可；或者两者都取，产生二维标准正态分布随机数。

在产生了标准正态分布  $X$  后，通过一个很简单的线性变换  $Y=X \delta + \mu$  即得到  $Y \sim N(\mu, \delta)$ 。

### 4. 离散随机变量采样的直接变换法

随机变量如果只取某些离散值，就称它为离散随机变量。典型的概率分布有：二项分布、几何分布、泊松分布以及负二项分布等。实现离散随机变量的采样常用直接变换法，在已知离散随机变量  $Y$  的分布列及概率累计值的条件下，用一个  $(0,1)$  均匀分布的随机数经过变换获得采样。它简要描述如下：

设随机变量  $Y$  分别以概率  $P_1, \dots, P_j, P_{j+1}, \dots, P_n$  取值  $Y_1, \dots, Y_j, Y_{j+1}, \dots, Y_n$ ，其中  $P_1 + P_2 + \dots + P_n = 1$ ，而  $Y_j$  对应的累计值如下定义

$$F_j = \sum_{i=1}^j P_i$$

注意这里没有必要先把  $P_n$  排序。

产生服从该分布列的随机数的步骤如下所述。

- 产生  $(0,1)$  间均匀分布的随机数，记为  $R$ ；
- 判断下式是否成立：

$$F_j \leq R \leq F_{j+1}$$

- 使上式成立的  $j$  值所确定的  $Y_j$  值即为随机变量的采样值。

直接变换法的优点是原理和操作都很简单，但应用于某些分布时计算量会很大，下面将介绍两个特殊的离散随机分布——泊松分布和二项分布随机数的产生方法。

#### 5. 泊松分布随机数的产生

泊松分布在排队论和军事运筹学中是一种常见的随机数。它的产生当然可以采用直接变换法，但其计算量会很大。这里不作证明地介绍一种简单的算法，对它的证明有兴趣的读者可以查阅参考文献。具体步骤为：

- (1) 初始化，置  $n=0$ ,  $p=1$ ;
- (2) 产生一个  $(0,1)$  间均匀分布的随机数  $Un$ ，用  $p \cdot Un$  替换  $p$ ;
- (3) 若  $p < e^{-a}$ ，则接受  $N=n$ ；否则拒绝，并将  $n$  增 1，然后返回步骤 (2)。

如果要产生 1000 个均值为  $a$  的泊松变量，则需重复上面的步骤 1000 次，一般而言，每产生一个泊松变量  $N$ ，平均需要  $a$  个随机数。如果  $a$  越大，该算法的效率就越低。

#### 6. 二项分布随机数

某一事件出现的概率为  $P$ ，则  $M$  个相互独立的相同事件中有  $n$  个事件出现的概率服从二项分布：

$$P(N = n) = C_m^n P^n (1 - P)^{m-n}$$

它的采样产生，如果使用直接变换法，计算量也会很大，其改进方法如下：

- (1) 初始化， $l=0$ ,  $n=0$
- (2) 产生  $(0,1)$  间随机数  $R$ ，如果  $R > (1-P)$ ，转步骤 (4)，否则顺序执行。
- (3)  $n=n+1$ ;
- (4)  $l=l+1$ ;

如果  $l > m$ ，输出  $N=n$ ，否则返回步骤 (2)。

最后在结束这一节之前，强调一点。无论是采用何种方法产生随机数，为了确保我们产生的随机数具有预想的统计特性，我们应该对所产生的随机数进行统计检验。对于不同的分布，都有相应的统计假设检验的方法，读者可以在相关的统计书籍里找到，这里就不再详述。此外，在上面所讲的种种方法中，有一个很重要的前提条件是独立性，为了慎重起见，独立性的检验同样是必不可少的。常用的检验方法有多维频率检验、连列表独立性检验、序列检验、相关系数检验、多维矩检验以及连法检验等，限于篇幅所限，这里也不再详述，读者如有兴趣可查阅有关著作。

## 1.4 输入数据的分析

输入数据是仿真实验的动力。在排队系统仿真中，典型的输入数据可以是到达的时间间隔和服务时间的分布。

要得到一个正确的输入数据模型要经过以下 4 个步骤：1) 收集原始数据；2) 基本统计分布的辨识，这通常根据数据的频数分布或直方图作出分布假设；3) 在上面的分布假设的前提下，估测分布的参数；4) 进行拟合度检验，以确定所假设的分布形式是否与收集到的数据

吻合，如果不吻合，就要转到第 2) 步，作出新的假设，重复上述过程。如果这个过程重复了若干次后仍不吻合，一般使用经典的分布形式。比如，通信信道的噪声通常假定成高斯白噪声。

### 1. 数据的收集

在用计算机仿真解决实际问题时，数据的收集是工作量最大的一项工作，也是最重要最困难的一个步骤。输入数据对模型正确的重要性是显而易见的。在数据收集时应做好以下几项工作：

- (1) 做好计划。从观察入手，尽量收集数据，为此要有耐心，千万不能伪造数据。
- (2) 收集时要注意分析。分清有用数据和无用数据，对无用数据没必要多采集。
- (3) 为了确定两个变量间是否存在相关性，要建立散布图，或者进行回归分析。
- (4) 考察一个似乎是独立的观察序列存在自相关的可能性。

### 2. 分布的辨识

这一步的目的是对所收集的数据的概率分布形式作出假设，以便进行后面的参数估计和分布的假设检验。

首先要画出频数分布或直方图。直方图的数据取值范围一般是等宽的，当然有时也用到不等宽的区间。在画直方图中，分组区间的组数依赖于观察次数以及数据的分散程度。经验的法则是选择样本量的平方根。对于连续数据的直方图，一般可以将每一区间频数中点连接起来便于观察。

至于直方图产生的方法，MATLAB 提供了方便的函数，它的使用方法将在下一章介绍。

画出直方图后，将直方图的形状和理论分布的形状比较，猜测其可能的分布，通常的可能有指数分布、正态分布和泊松分布。

### 3. 参数估计

为了将假设分布转化为指定分布，同时为了检验假设结果，需要对分布的参数作数值估计。表 1-1 列出了一些常用分布的参数估计量。

表 1-1

参数估计量

分 布	参 数	建议使用的估计量
泊松	$a$	$\hat{a} = \bar{X}$
指数	$\lambda$	$\hat{\lambda} = 1/\bar{X}$
在 $(0, b)$ 上的均匀分布	$b$	$\hat{b} = (n+1)/(n\bar{X}_{\max})$
正态	$\mu, \sigma^2$	$\hat{\mu} = \bar{X}, \hat{\sigma}^2 = S^2$

在进行参数估计时，样本均值和样本方差是两个经常使用的统计量。如果样本量为  $n$ ， $n$  个观测值为  $X_1, X_2, X_3, \dots, X_n$ ，则有以下定义

样本均值

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{n}$$

样本方差

$$S^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n-1}$$

如果离散数据已按频数分好组，则

$$\bar{X} = \frac{\sum_{j=1}^n f_j X_j}{n}$$

$$S^2 = \frac{\sum_{j=1}^n (f_j X_j - n\bar{X})^2}{n-1}$$

要注意一点，式中的频度  $f_j$  指采样值落入区间  $j$  的累计次数，并没有归一化，也就是所有的频度之和不是 1，而是采样值的个数。 $X_j$  是每个分组的代表值。

#### 4. 拟合度检验

我们在对样本的分布作假设时，主要的依据是主观的判断猜测。在实际中，并不是所有的输入数据都是很理想很完美地服从某个特定的理论分布的，而且不同的理论分布间的形状有时也很接近。读者在判断时，往往会觉得既像这种分布又像那种分布。因此必须对所做的假设进行定量的检验，常用的方法是  $\chi^2$  平方检验。这种检验方法首先是把  $n$  个观察值分成  $k$  个分组区间或单元。检验的统计量由下式给出

$$\chi_0^2 = \sum_{i=1}^n \frac{(o_i - e_i)^2}{e_i}$$

式中， $o_i$  是指第  $i$  个分组区间的观察频数， $e_i$  则是相应区间的期望频数，它与你所假设的分布有关，可以通过总的采样数  $n$  和该区间的理论计算概率的乘积获得。

可以证明  $\chi_0^2$  近似服从具有自由度  $f = k - s - 1$  的  $\chi^2$  分布， $s$  表示假设分布的参数个数，如泊松分布只有一个，而正态分布（又称高斯分布）有两个参数。自由度  $f$  在查表时很有用。

我们的假设检验描述为

H0: 观察值  $X_i$  是一组属于分布函数  $F$  的独立同分布的随机变量；

H1: 观察值  $X_i$  不是一组属于分布函数  $F$  的独立同分布的随机变量。

在假设 H0 成立的前提下，一般  $\chi_0^2$  的值会比较小；反之如果它的值太大，假设 H0 成立的可能性就不太大，这里就存在一个临界值问题。在假设检验时，通常要给定一个显著性水平  $\alpha$ ，根据  $\alpha$  和自由度  $f$  可以在表中查出临界值（一般统计教科书中都有  $\chi^2$  表），如果实际计算出的统计量大于查出的临界值，则表明你所做的假设不正确。

在应用这个检验时，如果期望的频数太小，将影响检验的有效性。根据经验，区间的个数在 40 以下，并能使所有区间的期望频数都大于 5，这样选择比较合适。如果某个区间的期望频数太小，就可以把这个区间和相邻区间合并，这时对应的观察区间也应按合并后的区间重新计算，自由度也要相应调整。

在输入数据的分析阶段,还应该对随机变量的相关性进行检验,这就要使用回归技术。关于回归技术的运用,许多统计书籍都有介绍,感兴趣的读者可以自行查阅。

一个科学的完整的仿真研究,还应包括对输出结果的分析。输出分析就是对仿真中所产生的数据进行分析,其目的在于预测一个系统的性能或比较两个甚至多个不同系统设计的性能。

当随机数发生器用作仿真模型的输入时,所得到的输出数据呈现随机变异。如果将一组仿真实验结果看成对某个量的一个估计值,那么输出分析就可以作为统计分析来估计方差,或确定出达到确定精度所需的实验次数。

仿真最重要的一个应用是对各种系统设计方案进行比较。在比较时,不能简单从实验的结果作出结论,科学的方法是进行统计分析,以确定观察到的差别是由设计方案的不同引起的,还是仅仅由模型固有的随机波动引起的。

仿真的输出分析和性能比较是一个比较复杂的过程,涉及许多概率统计知识,讨论这些理论并不是本书的主要目的。与其进行一个支离破碎的粗略介绍,还不如让读者在专门的书籍中寻找完整而又清晰的解答。

## 1.5 离散系统仿真

### 1.5.1 离散系统概述

若系统中状态的变化是在某些离散点或量化区间上发生,这样的模型称为离散事件模型,对应的系统称为离散事件系统。客观现实中,这样的系统是大量存在的。它不仅存在于工程系统之中,而且还大量出现于非工程系统(如经济、社会和生物)领域之中。例如,市场贸易、库存管理、设备装修、人口控制和交通管理等系统。在这种系统的研究、开发、设计和规划等工作中,人们经常需要了解哪些是系统的控制因素以及它们对系统稳定性和发展进程等方面的影响。例如,在经济领域中,商品的价格可能会在平均价格水平上发生周期性波动或者临时出现对商品价格的控制失灵。人们都希望能够预测这些现象并能够对它们进行控制。

#### 1. 离散系统的基本要素

离散系统的基本要素主要有:

(1) 实体。一般指系统所研究的对象。用系统的术语说,它是系统边界内的对象,系统中流动的或活动的元素都可以成为实体。例如,企业生产经营系统中,采购部、北京时间、制造车间和销售部门等均可称为实体。

(2) 属性。实体由它的属性来描述,属性反映实体的某些性质。例如,在商店系统中的顾客是一个实体,性别、年龄、优先权、购物时间、排队时间、服务时间以及购物费用等便是它的属性。

(3) 时刻。在系统的某个时间数值上,至少有一个实体的属性被改变,则称此时间数值为时刻。

(4) 间隔。相邻两个时刻之间的持续时间称为间隔。

(5) 状态。在某一个确定时刻,对系统实体和属性的描述称为状态。

(6) 事件。在某个时刻,系统状态变化的产生,称为一个事件。“事件”是改变系统状



态的实体的瞬间行为。例如，在商店中，顾客的到达或离去，对顾客服务的开始或结束，实体的产生或消失等都是事件。

(7) 活动。实体的一个持续期间称为活动。活动的开始或结束的瞬间则是一事件。

(8) 进程。进程由若干事件及若干活动组成，一个进程描述了它所包括的事件及活动间的相互逻辑关系及时序关系。如，一个顾客到达系统，经过排队，直到服务员为其服务完毕后离去可称为一个进程。

## 2. 离散系统仿真模型的结构

离散系统的模拟模型，一般由以下几个部分组成：

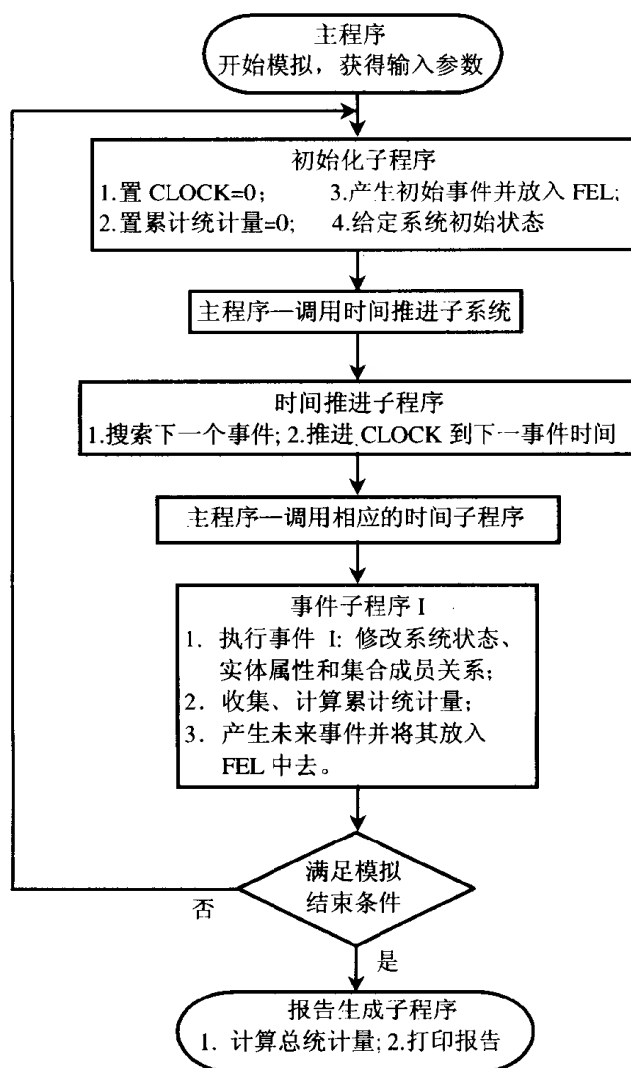


图 1-4 离散系统仿真模型结构图

(1) 系统状态。它是由一组系统状态变量构成，用它来描述系统在不同时刻的状态。

(2) 模拟时钟。用来提供模拟模型当前时刻的变量，它描述系统内部的时间变化。

(3) 时间表。在模拟过程中按时间顺序所发生的事件类型和时间对应关系的表，称为时间表。

(4) 统计器。在计算机仿真中往往设计一些工作单元，用来控制与存储仿真过程中结果

的统计信息，并进行统计计数，这些工作单元就叫作统计器。

(5) 定时子程序。该子程序可依据时间表来确定下一个事件，并将模拟时钟推移到下一个事件的发生时间。

(6) 初始子程序。在开始仿真时对系统进行初始化的子程序。

(7) 事件子程序。一个事件子程序对应于一种类型的事件，它在相应事件发生时，就转入该事件子程序进行处理，更新系统状态。

(8) 仿真报告子程序。它在仿真结束时，用来计算与打印仿真结果。

(9) 主程序。它用来调用定时子程序，控制整个系统的仿真过程，并确定下一事件，传递控制给各事件子程序以更新系统状态。

综上所述，离散系统仿真模型的结构如图 1-4 所示。图中 FEL 为“未来事件表”(Future-Events List)，其中存放表示所有当前正在进行的活动的结束事件。表中每一个事件记录至少包括两项：事件时间和事件类型。

### 3. 模拟时钟

在仿真程序中使用的时钟（计数器），其单位可以是微秒、秒、分、时、天、月和年等。一般，仿真运行开始时，模拟时钟初值为 0。

模拟时钟和实际时钟的区别在于：前者是离散的，后者是连续的。模拟时钟的读数应与现实系统中事件发生的时间一致，但模拟时钟的时间决不等于计算机进行模拟运行的时间。仿真运行时间完全取决于被仿真系统的特点、模型复杂的程度，以及模拟时钟所取单位的大小。例如，当模拟一个经济系统时，模拟时钟的单位可以取月或年。这样，即使现实系统运行了几年甚至几百年，而高速计算机用于仿真的运行可以在几分钟之内就完成。

## 1.5.2 离散系统仿真的基本方法

### 1. 离散系统仿真的基本策略

离散系统仿真的基本策略有三种，分别是事件调度法 (Event Scheduling)、活动扫描法 (Activity Scanning) 和进程交互法 (Process Interaction)。

#### (1) 事件调度法

离散系统中最基本的概念是事件。用事件的观点来分析真实系统，通过定义事件及每个事件发生引起系统状态的变化，按时间顺序确定并执行每个事件发生时相关的逻辑事件，这就是事件调度法的基本思想。

按这种策略建立模型时，所有事件均放在事件表中。模型中设有一个时间控制成分，该成分从事件表中选择具有最早发生时间的事件，并将模拟时钟修改到该事件发生的时间，再调用与该事件相应的事件处理模块，该事件处理完后返回时间控制成分。这样，事件的选择与处理不断地进行，直到仿真终止的条件满足或程序事件产生为止。

#### (2) 活动扫描法

有时，事件发生不仅与时间有关，而且与其他条件有关，即只有满足某些条件时才会发生。在这种情况下，采用事件调度法就显示出这种策略的弱点。原因在于，这类系统的活动持续时间具有不确定性，因而无法预定活动的开始或终止时间。

活动扫描法是针对具有上述特点的系统产生的。这种策略的基本思想是：系统由成分组成，而成分包含着活动，这些活动必须满足某些条件；每一个成分均有一个相应的活动子进

程；仿真过程中，活动的发生时间也作为条件之一，而且较其他条件具有更高的优先权。显然，活动扫描法由于包括了对事件发生时间的扫描，而具有事件调度法的功能。

### (3) 进程交互法

离散系统仿真的第三种方法是进程交互法。我们知道，一个进程包含若干个有序事件及有序活动。进程交互法采用进程描述系统，将模型中的主动成分历经系统时所发生的事件及活动按时间顺序进行组合，从而形成进程。一个成分一旦进入进程，它将完成该进程的全部活动。进程交互法既可预定事件，又可对条件求值，因而它兼有事件调度法及活动扫描法两者的特点。

上述三种基本策略各有优缺点，在离散系统仿真中均得到广泛的应用。事件调度法建模灵活，可应用范围广泛，但一般要求用户用通用的高级语言编写事件处理子例程，建模工作量大。活动扫描法对于各成分相关性很强的系统来说模型执行效率高。但是，用户建模时，除了要对各成分的活动进行建模外，仿真执行程序结构比较复杂，其流程控制要十分小心。进程交互法是建模较为直观的策略，其模型表示接近实际系统，特别适用于活动可以预测、顺序比较确定的系统，但是其流程控制复杂，建模灵活性不如事件调度法。

## 2. 模拟时钟的推进

离散系统的大部分属于动态系统，即与时间有关系的系统。因此，在进行系统仿真时需要考虑如何描述系统的时间变化，即时间如何推进。这里介绍两种基本方法：一种叫做周期扫描法，一种叫做事件扫描法。

### (1) 周期扫描法

这种方法是使模拟时钟按固定时间步长（必须足够小）向前推进，每推进一步，就扫描一次全部临近的未来事件产生时刻和产生条件，看产生时刻是否小于或等于当前时刻和有无产生条件已得到满足的事件，如果有，则仿真该事件；否则，就继续向前推进模拟时钟，如此不断地重复下去。周期扫描法的处理步骤如下：

- a. 初始化，模拟时钟  $CLOCK$ 、系统状态及统计量等置以初始值；
- b. 时钟  $CLOCK$  增加步长  $\Delta T$ ；
- c. 扫描事件表，若此刻无事件发生，则转步骤 b，否则处理该事件，并相应地改变系统状态；
- d. 收集统计数据；
- e. 若仿真终止条件未满足，则返回步骤 b，否则，执行下一步；
- f. 分析收集的统计数据，产生报告。

在周期扫描法中，在间隔内发生的所有事件都被当作是在间隔的末端所发生的。因此，本方法的缺点是在时间上间隔较小的事件却表现为同时发生。为克服这个缺点，需要将时间步长取得足够小，但步长越小，完成仿真所需的计算量就越大。周期扫描法的另一个缺点是每个时间间隔给予了同等重要的注意，而实际上，在没有事件发生的时间间隔里，并没有做什么有用的事件，只是模拟时钟的推进。如果系统存在相对长的非活动时期，而只有较短的活动时期，这样就使得大部分仿真运行时间只是单纯的推进时钟，从而造成很大的浪费。

因此，一般很少采用周期扫描法。但有些特殊情况却适于采用这种方法。例如，事件以规则的方式出现时，或进行某些大型系统仿真，而其中的各随机变量间的关系并不明确，或无已知关系时，采用周期扫描法则比较合适。

## (2) 事件扫描法

此法的模拟时钟的推进，是按照下一事件的发生时刻来触发的。大多数发生的事件，并不是某个固定的、预先确定的间隔。两个相邻事件发生的时间间隔一般是随机的，因此事件扫描法是一种变步长法。采用事件扫描法时，当事件被发现或产生时，它们是按时间的先后次序排列在一个表格里，时钟推进间隔的长度只由扫描事件表里下一个最早发生的事件所确定。因此，模拟时钟是按被仿真的事件的发生时间推进的。事件扫描法的处理步骤如下：

- a. 初始化，模拟时钟 **CLOCK**、系统状态及统计量等置以初始值；
- b. 扫描事件表，时钟 **CLOCK** 增加到下一个最早发生事件的时间上；
- c. 处理该事件，相应地改变系统状态；
- d. 收集统计数据；
- e. 若仿真终止条件未满足，返回步骤 b，否则，执行下一步；
- f. 分析收集的统计数据，产生报告。

事件扫描法目前被广泛采用，一般情况下它省时间，但事件扫描法的实现和设计比较复杂，而周期扫描法则具有较易实现和简单的优点。

## 第二章 用 MATLAB 实现静态仿真

### 2.1 MATLAB 基础

这一节，是专门为那些正在为选择何种语言编写仿真程序而烦恼的读者而写的，MATLAB 对这些读者而言是一种全新的语言。在这一节，我会向读者展示 MATLAB 非凡的功能，看完这一节，相信读者会由衷地觉得选择 MATLAB 编写仿真程序的确是一个蛮不错的决定。

#### 2.1.1 为什么选用 MATLAB

MATLAB 的初学者在开始接触 MATLAB 时，一定会有许多的疑问。例如，MATLAB 是什么，干什么用的，有什么好处等等。关于这些问题的回答，也许亲身的体验比严密、精确的论述的说服力要强得多。

请读者双击桌面上的 MATLAB6.0 图标，启动 MATLAB。这时候出现一个标题为“MATLAB Command Window”的窗体，这就是 MATLAB 的主界面，用过 VC++ 或 C 的人可能会觉得这个界面比较简单。正如这个界面一样，MATLAB 的使用也是十分简单方便的。对于安装最新版本 MATLAB6.0 的读者，MATLAB 的开发环境可能要变得复杂一些，图 2-1 显示的就是 MATLAB6.0 的开发桌面。这个桌面把 MATLAB5.3 以前的命令窗口变成了系统的一个子窗口，并新添了一些子窗口，而且增加了很多的图形操作，但总的来说和以前的版本变化不大，读者可以自己探索各个窗口的使用。

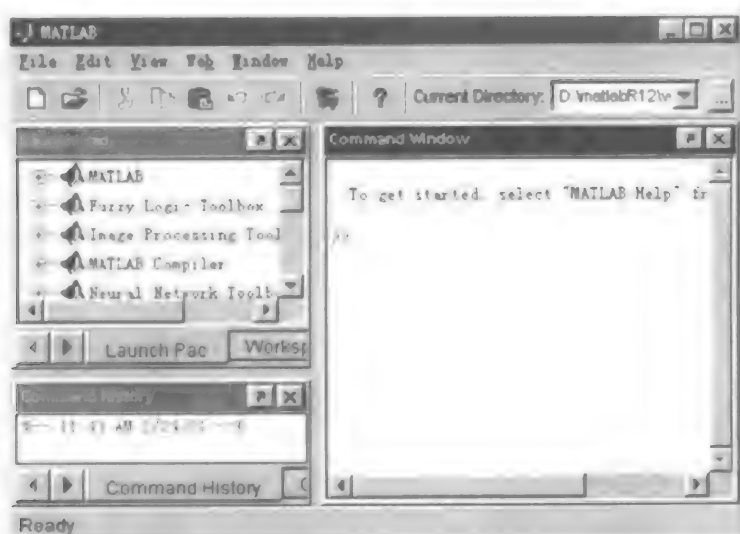


图 2-1 MATLAB6.0 开发环境

细心的读者可能已经注意到了窗体上面的那段话（仅对 5.3 版本而言），你可以分别敲入这些命令试试会有什么结果。例如，在命令窗口中输入

```
>> demo
```

回车之后，会出现如图 2-2 所示的窗口，它里面有许多 MATLAB 提供的演示程序，这些程序基本上体现了 MATLAB 应用的概貌。它们大都很精致，也比较有趣，读者可以自己探索，相信会带给你许多惊喜。至于 `helpwin` 和 `helpdesk` 这两个命令都是用来启动 MATLAB 的帮助窗口的，尤其是 `helpdesk`，里面汇集了 MATLAB 几乎所有的帮助文档，你可以在里面找到任何想要的信息，但可惜是用英文书写的，这对许多很厌烦英语的人是个障碍。除了语言上的原因外，帮助文档由于是手册似的编写体制，所以就显得过于繁杂。得益于次，这才诞生了许多关于 MATLAB 的书籍，本书正是其中之一。

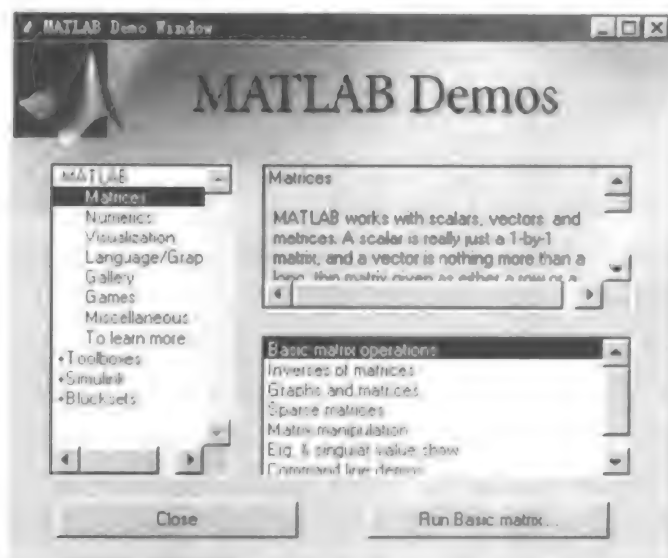


图 2-2 MATLAB 的 Demo 窗体

MATLAB 中的许多命令除了可以用上面的那种命令行形式运行外，也可以通过菜单命令来实现，比如上面的几个命令都可以找到相应的菜单命令，读者不妨自己试试。

MATLAB 的一个基本特性是其演草纸式的数学运算功能，它的意思是说你可以像在计算器上一样进行算术运算。例如，你可以输入

```
>> 4+6+2
```

```
ans = 12
```

这里需说明一点，“>>”并不是需要输入的内容，只是本书用来区分输入的命令与输出的结果之间区别的一种符号。有时候这一点也可以通过 `ans` 来区分，但这不是绝对的。对于 MATLAB6.0 就不存在这个问题了。

MATLAB 最吸引人的地方是什么呢？答案是它的矩阵运算功能。而 MATLAB 名称的得来也就是根据这一点——**Matrix Laboratory**。而矩阵运算如果用 FORTRAN 或 C 来实现，那将会是一件很令人烦恼的事情。你不但需要对所用的有关算法有深刻的了解，还需要熟练掌握

握所用语言的语法和编程技巧。往往是经过艰苦卓绝的写代码并调试成功后，所写的百余条甚至几百条代码仅仅是实现了一个矩阵的求逆工作，显而易见这样的编程效率极其低下。然而在 **MATLAB** 里实现一个矩阵的求逆，你所需做的事情是极其简单的。

首先我们来定义一个矩阵

```
>> A = [1 2 3; 4 5 6; 7 8 0] % 这样就定义了一个矩阵变量 A
```

```
A =
```

```
1     2     3
```

```
4     5     6
```

```
7     8     0
```

```
>> inv (A) % 求矩阵 A 的逆
```

```
ans =
```

```
-1.7778    0.8889   -0.1111
```

```
1.5556   -0.7778    0.2222
```

```
-0.1111    0.2222   -0.1111
```

是不是很简单？这个例子首先定义了一个矩阵 A，然后再求出它的逆矩阵。在 **MATLAB** 里定义一个变量是很方便的，无需进行类型说明。我们再举一例，如果要取出上述矩阵 A 的第一行，在高级语言里实现这一点恐怕要用到循环语句，但在 **MATLAB** 里是再简单不过的。

```
>> A (1,:)
```

```
ans =
```

```
1     2     3
```

通过上面的两个小例子，你应该对 **MATLAB** 方便、快捷、灵活的矩阵运算功能有了初步的体会。正因为此，它才会如此受到专业研究人员的广泛重视，在工程计算和仿真有着广泛应用。

概括地说，与 C、C++、FORTRAN 这类高级程序设计语言相比，**MATLAB** 不但在数学语言的表达上与解释方面表现出人机交互的高度一致，而且具有作为优秀高技术计算环境所不可缺少的如下特征：

- (1) 高质量、高可靠的数值计算能力；
- (2) 基于向量、矩阵和数组的高级程序设计语言；
- (3) 高级图形和可视化数据处理能力；
- (4) 广泛解决各学科专业领域内复杂问题的能力；
- (5) 拥有一个强大的动态系统仿真建模工具箱——**Simulink**；
- (6) 支持科学和工程计算标准的开放式、可扩充结构；
- (7) 跨平台兼容。

这里 **Simulink** 是本书的主要论述对象。

MATLAB 的强大吸引力还表现在其有许多功能强大，专业化的实用工具箱。例如，自动控制、图像信号处理和信号分析等领域都有相应的工具箱。

熟悉 MATLAB 的人，都会有这样的感受，MATLAB 更像是一个函数的海洋，各种各样的应用领域都可以在 MATLAB 里找到对应的函数(这可以减少你投入常用算法设计的精力)，尤其是有了工具箱之后，使函数的功能更加专业化。这么多的函数，是不是要一个一个掌握呢？完全没有必要，也不可能。许多书籍将 MATLAB 的函数一个个介绍过来，逐个学习这些函数固然很重要，但不是最有效的方法。

学习 MATLAB 的一个比较好的策略是即用即学。这就要求读者要掌握 MATLAB 语言的基本特性，并具备自己找到所需要的函数和它的使用语法的能力。下面的章节，将按照这个思路来进行论述。

### 2.1.2 MATLAB 基本特性

MATLAB 的一些最基本的概念读者必须要熟悉。

#### 1. MATLAB 的工作空间

工作空间是存储变量和命令的区域，如果你的命令是用命令行形式(如前面一小节所述)或 M 文件的(后面将会讲到)形式输入，那么输入的命令和创建的所有变量值，就会驻留在 MATLAB 的工作空间，可以在任何需要的时候调用。紧接前面的例子，变量 A 已经被创建，可以重新查询它的值。

```
>> A
```

```
A =
```

```
1     2     3
4     5     6
7     8     0
```

MATLAB 提供了一些命令用于管理工作空间。

如果忘记了变量名，可以用 who 命令来查询变量。

```
>> who
```

```
Your variables are:
```

```
A
```

请注意，MATLAB 是区分大小写的，MATLAB 里的命令或函数名称都是小写。

MATLAB 里用于工作空间管理的命令还有 clear 和 whos。前者用于清除变量，后者与 who 的作用相近，只是还列出变量的大小和数据类型。读者可以自己尝试一下，这些命令后面有特定的变量名作参数，则只作用于该变量，否则指整个工作空间的所有变量。clear 命令的使用一定要谨慎，变量一经它删除就无法还原。

而在 MATLAB6.0 里，读者可以在 workspace 子窗口查询工作空间，而它的 command history 子窗口则显示了以前输入的命令记录。



2. 关于变量和函数的命名

这是学习任何语言都必须了解的。MATLAB 也有自己的命名规则，有些和其他的高级语言一样，有些不一样。表 2-1 是 MATLAB 详细的命名规则。

表 2-1 变量命名规则

变量（函数）命名规则	说明/举例
变量名（函数名）区分大小写	Test, test, TEST 等不是相同的变量名
变量名（函数名）最多不能超过 19 个字符；超出的字符将被忽略	
变量名必须以字母开头，之后可以是任何字母、数字或下划线。 由于许多标点符号在 MATLAB 中有特殊含义，所以变量名（函数名）不允许使用	How_about X51484 A_b_c_d_e

此外，MATLAB 还定义了一些特殊变量，表 2-2 是其中常用的一些。

表 2-2 常用特殊变量

特殊变量	作用
ans	用于返回结果的缺省变量名
Pi	圆周率
i 和 j	虚数的表示符号
nargin	所用函数的输入变量
nargout	所用函数的输出变量
flops	计算机的最小数，与 1 相加就产生一个比 1 大的数

用户可以对特殊变量重新定义，但最好不要这样。

3. 注释和标点

一行中，“%”后面的所有文字都是注释，MATLAB 对中文注释支持好像不大好，本书的许多注释，都是后来加上的，读者在试验时可以把它们去掉。例如

>> who %查询变量列表

多条命令可以放在同一行，只要它们被逗号或分号隔开。

>> a=1, b = 2; c =3

a =

1

c =

3

为什么没显示 b？逗号或没有标点符号都代表把结果显示在命令窗口，而分号禁止显示。MATLAB 里分号的作用和 PASCAL、C 等高级语言不一样，要注意区分。

连续三个点“...”表示语句的余下部分将在下一行出现，但变量名不能被两行分开，同样注释语句也不能续行。

在任何时候，键入 Ctrl + C 都会中断 MATLAB 的当前执行内容（有关其他在计算机平台

上中断 MATLAB 的信息请看帮助)。

#### 4. MATLAB 的搜索路径

当你在命令行输入一个字符串 (有可能是变量也有可能是命令), 回车后, MATLAB 按一定的顺序来搜索执行。例如, 输入 ptest

```
>> ptest
```

MATLAB 对它进行解释的搜索顺序为:

- (1) 检查 ptest 是否为 MATLAB 工作空间中的变量;
- (2) 检查是否为内置函数;
- (3) 检查 MEX 文件 ptest.mex 是否存在于当前目录;
- (4) 检查 M 文件 ptest.m 是否存在于当前工作目录;
- (5) 按次序搜索已设置的路径, 检查 ptest.mex 或 ptest.m 是否存在于 MATLAB 的搜索路径中。

MATLAB 一找到匹配的文件, 就执行它。在 MATLAB 里存储文件或导出文件的顺序也是从当前目录开始, 然后才是搜索路径中的目录。例如

```
>> save 'filename.mat' A % 把变量保存在一个数据文件里
```

由于你的命令里没有指明文件的绝对路径, 所以 MATLAB 的缺省处理是把它生成在此时的当前目录里。所以运行此类命令时, 最好是先设置好当前目录 (桌面的右上端, 最好在设置路径时预先设置), 或者在使用命令时把路径写好, 免得出现不必要的麻烦。而且, 最好把工作路径设为路径, 就像输入 MATLAB 命令一样, 可以直接命令窗口输入自己的函数。也可以使用命令行命令进行这些设置, 如 path 命令。但 File 菜单下的 Set path 命令似乎更直观。点击这个菜单项后会出现一个名为 Path Browser 窗体, 用户可以在此设置和浏览路径。设置路径的做法是: 先通过浏览按钮选择你设置为路径的目录, 然后通过 path 菜单下的 Add to path 命令设置为路径, 最后别忘了保存你刚才的设置, 这个命令在 File 菜单下, 之后关闭这个窗口就可以了。

设置搜索路径, 可以使用户自己编写的 M 文件或 M 文件函数, 直接可以在命令行直接输入来调用。

#### 2.1.3 MATLAB 的三种执行方式

MATLAB 里除了直接在命令行输入命令的方式来进行计算外, 还有两种方式可以使用, 即 M 文件和 M 文件函数。

通过在命令行输入命令的方式快速有效, 交互性很好, 这仅仅是在问题简单时如此。一旦命令数增加或者期望改变一个变量的值, 这种方式就显得非常复杂了。比较好的替代方式是脚本文件和 M 文件函数。它们使用户具备进行更复杂的计算的能力, 这两种文件的扩展名都是.m。

##### 1. 脚本文件

脚本文件是 MATLAB 为解决上述问题, 提供的一种逻辑解决方案。所谓脚本文件, 有时简称为 M 文件, 其实就是一个批处理文件, 它允许用户把命令放在一个简单的文本文件中, 然后告诉 MATLAB 打开文件并执行命令, 就如同在 MATLAB 命令行输入命令一样。而且你

还可以进行复杂的程序设计。

在体验如何用 MATLAB 语言进行程序设计之前，我想提醒读者先用 File 菜单下的 Set Path 命令设置好路径，并把当前目录置为你的工作目录，这样可以为以后的工作省去不少麻烦。

要编辑 M 文件，必须启动文本编辑器。这你可以用 File/NewM-file 命令来完成这项工作。这时会出现一个标题为“MATLAB Edit/Debug”的窗体。在这里你就可以编写 M 文件和 M 文件函数。

以前面的矩阵求逆例子为例，你可以这样写

```
%这是一个求线性方程解的 M 文件示例
b = [1; 2; 3];
A = [1 2 3; 4 5 6; 7 8 0];
Inv(A)*b;
```

写完之后，请保存这个文件，文件名可以是任何你喜欢的名字比如 example1，但存储的目录我建议选你已经设置过路径的目录，在如果读者已把那个目录设为了当前目录，保存时的缺省目录就是当前目录。运行它的方式为

```
>> example1
```

如果你觉得这样转来转去执行函数不方便，在 MATLAB 编辑窗口的 Tools 菜单有一个 Run 命令可以一定程度的满足你的要求。这种方式在调用需要参数的 M 文件函数时，就不管用了。

这时你会发现在命令窗口什么都没有出现。用 who 命令查询变量列表，就会发现多了几个变量：b、A 和 ans。这便是使用分号的效果，而在大多数高级语言里分号意味着一条语句的结束，是必不可少的。

这里可以得出一个结论，脚本文件的工作空间和 MATLAB 的工作空间是相同的，这和后面将介绍的 M 文件函数有很大的不同。

M 文件的另外一个用途，是用来输入大的数组，通过文本编辑器输入一个或多个数组可以很容易修改错误，不需重新输入数组。这一点在仿真应用中很重要。

M 文件里的命令是不间断地批量执行的，这样与用户的交互能力就丧失了，为此 MATLAB 提供了一些函数专门用来进行这种交互。本书的附录列出了它们，详细的用法可以参考 MATLAB 帮助文档。

## 2. M 文件函数

我们还是以上面求线性方程解的例子来说明 M 文件函数。则要作如下的改动

```
function s=example2(A,b)
%这个函数求线性方程组 As=b 的解 s.
if det(A)~=0
    s= inv (A)*b;
```

```
else
    error ('A 是一个奇异矩阵')
end;
```

保存 M 文件时要注意，文件名和函数名必须相同，在这里就是 `example2`。道理很简单，因为执行时是按文件名是否匹配来进行的，这样用户自己写的 M 文件函数就可以像命令一样使用了（当然它要被保存到读者已添加到搜索路径中的目录）。

运行程序，请回到命令窗口

```
>>s=example2(A,b)
```

```
s =
   -0.3333
    0.6667
         0
```

M 文件函数和脚本文件相似的地方都是以后缀名 `.m` 结尾，但在机制上有很大的差别。M 文件函数有自己专用的工作空间，它与 MATLAB 的工作空间分开。函数内部变量与 MATLAB 工作空间之间的唯一联系是函数的输入和输出变量，这与 M 文件完全不同。而且函数的参数是值传递的，换言之，你在函数内部对输入变量做任何修改，都不会影响 MATLAB 工作空间的变量。

下面就 M 文件函数归纳几点：

（1）函数名和文件名必须相同，如上例中函数 `example2` 保存在 `example2.m` 文件中。

（2）在 M 文件函数中，到第一个非注释行为止的所有注释行是帮助文本。当需要帮助时，返回该文本。例如 `>>help example2`，返回第一行，MATLAB 对中文支持不好，注释最好是用英文。

（3）函数可以有零个或多个输入量。函数可以有零个或多个输出参量。调用时，输入变量和输出变量可以少于规定值，但不能超出。对于输入变量，为了保证函数正常的运行，你必须对变量少于规定值的情况进行特殊的处理。`nargin` 这个函数空间变量包含了实际输入参数的个数。在 `example2` 中减少输入参数，函数不能正常运行。

（4）当函数有一个以上变量，输出变量包含在方括号内。例如 `[v,d]=eig(A)`。

（5）如果 MATLAB 的特殊变量，例如 `ans`，在函数的工作空间中重新定义，不会延伸到 MATLAB 的工作空间，这就像高级语言的局部变量一样；反之也是这样。

（6）MATLAB 搜寻脚本文件和搜寻 M 文件函数的方式是一样的。比如 `>>example2`，它的搜索顺序如下：变量（即在 MATLAB 的工作空间）→内置函数→当前工作目录→MATLAB 搜寻路径上的所有目录或文件夹。这就是前面为什么先把你的工作目录加入到路径中去的原因。

（7）函数可以递归调用，即 M 文件函数可以调用它们本身。

（8）M 文件函数到达 M 文件终点，或者碰到返回命令 `return`，就结束执行和返回。`return` 命令提供了一种结束一个函数的简单办法，而不必到达文件的终点。

(9) 你可以用 `error` 函数在命令窗口显示一个字符串，放弃函数执行，把控制权交还给键盘。这个函数对提示函数使用不当很有用，如 `example2` 中输入的 `A` 没有逆矩阵时，就提示“矩阵 `A` 没有逆矩阵”。

MATLAB 的文件编辑器没有编译这个功能，其实 MATLAB 语言是一种脚本语言，是一种解释性的语言，不需先经过编译。那如何查找语法错误呢？一个技巧是设置断点，对语法上通不过的语句，会出现一个错误提示窗口，用户就可以相应的改正了。

最后还想强调一下 MATLAB 的全局变量，函数可以与其他函数、MATLAB 工作空间和递归调用本身共享全局变量。为此要在所有要共享该变量的工作空间把它设成共享，说明一个变量为共享的方法是“`global 变量名`”。下面的例子说明如何设置全局变量。

```
function tic
%TIC Start a stopwatch timer.
%   The sequence of commands
%       TIC, operation, TOC
%   prints the number of seconds required for the operation.
%
%   See also TOC, CLOCK, ETIME, CPUTIME.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.5 $   $Date: 1997/11/21 23:48:09 $

% TIC simply stores CLOCK in a global variable.
global TICTOC
TICTOC = clock;

function t = toc
%TOC Read the stopwatch timer.
%   TOC, by itself, prints the elapsed time (in seconds) since TIC was used.
%   t = TOC; saves the elapsed time in t, instead of printing it out.

%   See also TIC, ETIME, CLOCK, CPUTIME.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.6 $   $Date: 1998/05/13 00:20:24 $

% TOC uses ETIME and the value of CLOCK saved by TIC.
global TICTOC
if isempty(TICTOC)
    error('You must call TIC before calling TOC.');
```

```

end
if nargout < 1
    elapsed_time = ctime(clock,TICTOC)
else
    t = etime(clock, TICTOC);
end

```

在上面的例子中，**TICTOC** 是一个全局变量，它在两个需要共享它的函数里都进行了全局变量的设置。这样函数 **tic** 和 **toc** 都可以访问该变量。这虽然带来了方便，但也容易导致混乱，所以使用全局变量时一定要慎重。

### 2.1.4 MATLAB 里的函数

这一小节不准备专门讨论某一个具体的函数，或者是某一类。当然知道一些基本的函数和运算符是必不可少的。本书的附录列出了 **MATLAB6.0** 常用函数简单说明，至于具体用法请看帮助文档。而和仿真应用密切相关的那些函数，本章后面的章节会详细介绍。

这里要涉及的是 **MATLAB** 函数的一些共性。

**MATLAB** 是基于向量、矩阵和数组的语言，它的主要数据结构就是这三种，当然数值类型有很多。而 **MATLAB** 的大部分函数都是面向这三类结构的，返回值也是如此。

首先，先区分一下向量、矩阵和数组的区别。同数学上的定义一样，向量是一维的，表现形式上就是一行或者一列；而矩阵则是二维的，这里要分清维数和矩阵的大小的区别，常用  $M \times N$  来表示（行数  $\times$  列数）；而数组则是多维的，向量和矩阵都属于数组的一种，但因为它们较常用，所以单独提出来。例如

```
>> ones (2, 2, 2)    %定义一个全为 1 的数组
```

```
ans(:, :, 1) =
```

```

1      1
1      1

```

```
ans(:, :, 2) =
```

```

1      1
1      1

```

它由两个矩阵组成，而矩阵的每一行或每一列都是一个向量。

一般而言，**MATLAB** 函数都是基于向量和矩阵的，这和高级语言里的基于标量的函数要区别开来。例如，余弦函数 **cos**

```
>> x=[1 0.1 pi 0.5]; %定义了一个向量变量 x
>> cos(x)
```

```
ans =
    0.5403    0.9950   -1.0000    0.8776
```

对一个向量（数组）求余弦，实际的操作是对向量（数组）的每个元素分别求余弦。像余弦这种从标量运算推广而来的函数对数组的处理都与此类似，它们大多数是一些数学函数，如 `Ln`（求自然对数），`abs`（求绝对值）等等。

然而，有些 MATLAB 函数本身只对向量有定义，这时候要推广到矩阵或数组情况，MATLAB 有相应的规定。例如 `sum` 函数，它是求某一行或某一列的所有元素的和，只对向量有定义。例如

```
>> x = [1 2 3 4 5 6];
>> sum(x)
```

```
ns =
    21
```

MATLAB 里的函数使用很灵活，函数的通用性很好，对参数的要求不像有些高级语言那样刻板。就像 `sum` 函数，你也可以输入列向量，至于向量的大小更是可以随意选择，在编写 M 文件函数时可以用函数 `length` 或 `size` 来确定数组的大小。还是来看看 `sum` 函数，在 MATLAB 里它的帮助信息为（通过 `help sum` 命令得到）：

---

#### SUM Sum of elements.

For vectors, `SUM(X)` is the sum of the elements of X. For matrices, `SUM(X)` is a row vector with the sum over each column. For N-D arrays, `SUM(X)` operates along the first nonsingleton dimension.

`SUM(X,DIM)` sums along the dimension DIM.

See also `PROD`, `CUMSUM`, `DIFF`.

---

（为了便于读者阅读，这里笔者把原始的帮助文档翻译为中文）

**SUM** 对每个向量的元素求和。

如果 X 是一个矩阵，`SUM(X)` 返回一个由矩阵各列向量元素的和组成的行向量。如果 X 是一个 D 维的数组，`SUM(X)` 按第一个非单元元素的维进行求和。

`SUM(X,DIM)` 按第 DIM 维求和。

这是一个很典型的 MATLAB 函数帮助，你在许多函数的帮助信息中会遇到类似的信息。正如前面所述 `sum` 函数是基于向量定义的，MATLAB 将它扩展成矩阵时，缺省的处理是对每一个列向量求和，再把和值用一个行向量返回。这通常也是其他 MATLAB 函数对此类情况的缺省处理，但不是绝对的，MATLAB 在许多函数里提供了一个参数 `DIM`，它允许用户定义成按行向量加和，或按数组的其他维求和。

例

```
>> x=[1 2 3;4 5 6;7 8 9]; %定义一个 3×3 的矩阵
>> sum(x,2) %沿 x 的第 2 维求和，即把行向量求和
```

```
ans =
```

```
6
15
24
```

一个数组的所有元素，都可以用一个下标来标识，矩阵要 2 个坐标，D 维数组则要 D 个坐标。则沿 DIM 维求和的意思就是一次求和中，只有第 DIM 个下标变化。在上例中，sum(x,2) 就是把行向量求和，sum(x,1) 和 sum(x) 相同。

对于多维数组的情况也是类似的，只是由于显示的方便，往往要拆成许多个数组来表示。但第一个非单值维的表述有些费解，其实和向量情况是一样的，比如行向量，也可以理解成一个特殊的矩阵（只有一行），这样按矩阵的处理方法，则返回值就没有变化。但根据非单值维的要求，行向量的第一维只有一个值，也就是单值维，而第二维才是第一个非单值维，于是就得到按行向量求和。但这个准则在使用 DIM 参数时就失效了。

第三类函数是本身就是基于数组定义的如求逆等等，这时候它的推广并没有太大的变化，请读者自己体会。

最后，重申一下 MATLAB 的基本数据结构是数组（包括向量，矩阵），无论是在调用还是编写 M 文件函数时，都要养成向量化的思考习惯。不要依旧按标量的想法，否则就没有利用到 MATLAB 的优势。

### 2.1.5 MATLAB 里的矩阵（数组）运算

许多高级语言的计算都是基于标量的，尽管它们也支持矩阵（数组）类型，但在这些语言里建立和处理数组极其不方便，通常需要使用多个循环语句。然而 MATLAB 里，矩阵运算是十分方便的。读者在前面已经 MATLAB 函数对矩阵运算的强大支持。所以如何灵活、快捷的操作矩阵，是学习 MATLAB 语言一个主要内容。

#### 1. 矩阵的生成

MATLAB 生成矩阵的方法是很直观的，只需要按照顺序输入就可以了。以向量为例：

```
>> x=[1 .1*pi .2*pi .3*pi .4*pi .5*pi .6*pi .7*pi .8*pi .9*pi pi]
% 生成一行 11 列的矩阵，
% .1 是小数 0.1 的缩写。
```

```
x =
```

```
Columns 1 through 7
```

```
1.0000    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
```

```
Columns 8 through 11
```

```
2.1991    2.5133    2.8274    3.1416
```



可见创建数组的方法是很简单的，数组用方括号来包含，里面的元素值用空格隔开（逗号也可以）。对于建立二维的数组（矩阵），也可以用直观方法来建立，只需将行与行间用分号隔开就可以。例如

```
>> y = [ 1 2 3 ; 4 5 6 ; 7 8 9 ];
```

```
y =
```

```

1     2     3
4     5     6
7     8     9
```

但对于 3 维以上的数组，用直观的方法建立就不太方便。而且在数据量太大的情况下这种方法就会很麻烦。MATLAB 提供了一些便捷的方法。表 2-3 列出了常用的几种方法。

**表 2-3** 简单数组的创建方法

用 法	说 明
$x = (\text{first} : \text{last})$	创建从初始元素（first）开始，加 1 计数，到 last 结束的行向量
$x = (\text{first} : \text{increment} : \text{last})$	创建从 first 开始，加增量（increment），到 last 结束的行向量
$x = \text{linspace}(\text{first}, \text{last}, n)$	创建从 first 开始，到 last 结束，有 n 个元素的行向量 x
$x = \text{logspace}(\text{first}, \text{last}, n)$	创建从 $10^{\text{first}}$ 开始，到 $10^{\text{last}}$ 结束，有 n 个元素的对数分隔行向量 x

下面就分别就这几种方法举例说明。

```
>> x = (1:5)
```

```
x =
```

```

1     2     3     4     5
```

```
>> x = (0:0.1:1)*pi
```

```
x =
```

```
Columns 1 through 7
```

```

0     0.3142     0.6283     0.9425     1.2566     1.5708     1.8850
```

```
Columns 8 through 11
```

```

2.1991     2.5133     2.8274     3.1416
```

```
>> x = linspace(0, pi, 11)
```

```
x =
```

```
Columns 1 through 7
```

```

0     0.3142     0.6283     0.9425     1.2566     1.5708     1.8850
```

```
Columns 8 through 11
```

```

2.1991     2.5133     2.8274     3.1416
```

```
>> x=logspace(0,2,11)
```

```
x =
```

```
Columns 1 through 7
```

```
1.0000    1.5849    2.5119    3.9811    6.3096   10.0000   15.8489
```

```
Columns 8 through 11
```

```
25.1189   39.8107   63.0957  100.0000
```

当要建立列向量时，用直接法，就要把列向量看成多行单列的数组。例如，

```
>> c=[1; 2; 3; 4; 5] % 直接法
```

```
c =
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

也可以用转置符号“'”把行向量转置为列向量，这样就可以用生成行向量的方法来生成列向量。例如，

```
>> c= (1:4)'
```

```
c =
```

```
1
```

```
2
```

```
3
```

```
4
```

⚠ 注意，' 算子其实是复数共轭转置——转置的结果还造成虚部符号的改变，MATLAB 提供了一个非复数转置共轭算子——“.'”，它只进行转置，并不对数组元素进行共轭运算。

## 2. 访问矩阵的元素

矩阵里的元素可以通过下标来访问，例如对于行向量（列向量） $x$ ， $x(1)$  就代表  $x$  的第一个元素，对于多维数组，则用多维下标来访问，这种用法和一般的高级语言是差不多的。但 MATLAB 还支持访问一块元素，可以用冒号表示。

```
>> x=([1:1:6]; (2:1:7))
```

```
x =
```

```
1     2     3     4     5     6
```

```
2     3     4     5     6     7
```

```
>> x(2,1:3) % 访问 x 中的一块元素
```

```
ans =
```

```
2    3    4
```

事实 MATLAB 里访问数组元素的方法是很灵活的，访问元素的下标持续可以由另外一个数组来编址。请读者仔细体会下面的几个例子。

```
>> x(1,3:-1:1)
```

```
ans =
```

```
3    2    1
```

```
>> x(:, 2: 2: 7) % 单独的冒号表示所有的行
```

```
ans =
```

```
2    4    6
3    5    7
```

```
>> x([5, 4, 7])
```

```
ans =
```

```
3    3    4
```

在上面最后一个例子中，向量[5, 4, 7]就是作为编址矩阵来访问数组的元素，按照编址向量的元素取出元素，并放在相应的位置。例如上面的命令分别把 x 的第 5、4、7 个元素作为新数组的第 1、2、3 个元素。

### 3. 数组的操作

MATLAB 里数组支持的运算是十分灵活的。它可以和标量进行运算，也可以在数组间按元素对元素进行处理。例如，

```
>> g=[1 2 3 4; 5 6 7 8];
```

```
>> g*2-1
```

```
ans =
```

```
1    3    5    7
9   11   13   15
```

```
>> h=[1 1 1 1; 2 2 2 2];
```

```
>> g.*h % 在常用的运算符前加一个点，表示数组间元素对元素的运算，即 ./ 和 ./ 是完全不同的%
运算。
```

```
ans =
```

1	2	3	4
10	12	14	16

下面就来介绍 MATLAB 为矩阵提供的强有力的操作方式，它支持对矩阵进行插入子块、提取子块和重排子块等操作。请看下面的例子。

```
>> a=[1 2 3;4 5 6;7 8 9] % 定义矩阵 a
```

```
a =
```

1	2	3
4	5	6
7	8	9

```
>> a(3,3)=0 % 将第 3 行，第 3 列的元素置为零
```

```
a =
```

1	2	3
4	5	6
7	8	0

下面的命令把第 2 行，第 6 列的元素置为 1，但由于 a 不足 6 列，所以执行时首先把矩阵增加到所需的规模，并以零填满，使矩阵依然是矩形。

```
>> a(2,6)=1 % 将第 2 行、第 6 列的元素置为 1
```

```
a =
```

1	2	3	0	0	0
4	5	6	0	0	1
7	8	0	0	0	0

```
>> a=[1 2 3;4 5 6;7 8 8] % 恢复矩阵 a 原来的数据
```

```
a =
```

1	2	3
4	5	6
7	8	8

下面的命令的作用是按逆序提取 a 的各行，形成新的矩阵 b。

```
>> b= a( 3 :-1:1 , 1:3)
```

```
b =
```

7	8	8
4	5	6

1      2      3

将矩阵合并的操作在 MATLAB 里是很简单的。例如下面的命令，将 **b** 中的第一列向量和第 3 列向量附加到矩阵 **a** 的右边形成新的矩阵 **c**。

```
>> c=[a b(:, [1 3])]
```

**c =**

1	2	3	7	8
4	5	6	4	6
7	8	8	1	3

```
>> b= a(:) % 依次提取 a 的每一列，将 a 拉伸为一个列向量赋给 b
```

**b =**

1  
4  
7  
2  
5  
8  
3  
6  
8

下面就来删除矩阵中的一块。

```
>> b=a % 将 b 的数据复原
```

**b =**

1	2	3
4	5	6
7	8	8

```
>> b(:,2)=[] % 将第 2 列置为空矩阵
```

**b =**

1	3
4	6
7	8

上面的命令通过把第 2 列赋值一个空矩阵，来把这一列去掉。当把矩阵的某一块设置为

空矩阵[]，这一块就被删除，原来的矩阵只保留剩余的部分。需要注意的是必须删除整个行或列使余下的部分依然是矩阵。当然读者可能会想，这和提取矩阵的一块其实是一样的，但在某些时候，上面所说的方法要快一些。

请接着看下面的示例。

```
>> b= a(:,[2 2 2 2])
```

b =

2	2	2	2
5	5	5	5
8	8	8	8

这个命令重复 4 次提取 a 中的第 2 列所有行产生 b。这是一个很常用的技巧，它最常用在复制向量生成矩阵。

```
>> b=[1 4 7]
```

b =

1	4	7
---	---	---

```
>> c=b([1 1 1 1],:)
```

c =

1	4	7
1	4	7
1	4	7
1	4	7

下面的例子就是这个技巧的应用，请考虑下面的数据：

```
>> d=[1 2 3 4; 5 6 7 8; 9 10 11 12]
```

d =

1	2	3	4
5	6	7	8
9	10	11	12

```
>> v=[2;4;8]
```

v =

2
4
8

在很多情况下需要执行  $v$  和  $D$  各列之间的数学运算。例如，将  $D$  的各列减  $v$ ，则有：

```
>> e=[d(:,1)-v d(:,2)-v d(:,3)-v d(:,4)-v]
```

```
e =
```

```

-1     0     1     2
 1     2     3     4
 1     2     3     4
```

但是这种方法写起来很麻烦，可以用更快的方法：

```
>> e=d-[v v v v];
```

但这种方法还不是最好的，最好的方法是：

```
>> c=size( d, 2) % 求出矩阵的列数
```

```
>> e=d-v(:, ones(1, c))
```

最后，想提醒读者，只要逻辑数组和欲编址的数组大小相同，**MATLAB6.0** 还支持由 0 和 1 构成的所谓的逻辑数组来对数组进行编址。此时，真（1）元素被保留，假（0）元素被删除。通常是用逻辑算子或者关系算子来创建逻辑矩阵的。

```
>> x=-3:3
```

```
x =
```

```

-3    -2    -1     0     1     2     3
```

```
>> abs(x)>1
```

```
ans =
```

```

 1     1     0     0     0     1     1
```

```
>> y=x(abs(x)>1) % MATLAB5.3 似乎不支持这种用法
```

```
y =
```

```

-3    -2     2     3
```

```
>> y=x ([1 1 1 1]) % 将 x 的第一个元素提取 4 次
```

```
y =
```

```

-3    -3    -3    -3
```

上面的命令把  $x$  的第一个元素提取 4 次。为什么不能把  $[1 \ 1 \ 1 \ 1]$  看成逻辑数组而执行得到保留前面 4 个元素的结果呢？这是因为数组的元素个数比  $x$  少，所以就不当成逻辑数组。如果数组里出现 0 元素，如  $[1 \ 0 \ 1 \ 1]$ ，那么  $x([1 \ 0 \ 1 \ 1])$  将是无法执行的命令，因为 0 不是一个有效数组下标。

```
>> x(abs(x)>1)=[] % 舍弃 x 中 abs(x)>1 的值
```

```
x =
```

```
    -1     0     1
```

总之，读者在使用时，抓住 MATLAB 灵活、便捷的矩阵运算特点，自觉的运用矩阵运算的思维（并行计算），就会在编程运用中事半功倍。

### 2.1.6 MATLAB 里的程序设计

这里主要介绍 MATLAB 里的关系和逻辑运算，以及它的控制流语句。而 MATLAB 编程中所用到的基本数学运算，程序设计的格式（M 文件函数和 M 文件），前面已经描述过它们的特点。

#### 1. 关系和逻辑运算

和别的语言一样，MATLAB 里的逻辑和关系运算的目的就是提供求解真假命题的答案，而这些命题往往用于控制命令的执行流程。作为所有关系和逻辑表达式的输入，MATLAB 把任何非零数值当作真，输出为 1，把零当作假，输出为 0。MATLAB 包含所有常用的比较，如小于（<），等于（=），小于或者等于（<=），大于（>），大于或等于（>=），等于（==），不等于（~=），以及逻辑操作：与（&），或（|）和非（~）。

MATLAB 的关系操作和逻辑操作能比较两个尺寸同样大小的数组，或用来比较一个数组和一个标量。在后一种情况，标量和数组中的每一个元素相比较，结果与数组大小一样。例如

```
>> a=1:9,tf=a>4
```

```
a =
```

```
    1     2     3     4     5     6     7     8     9
```

```
tf =
```

```
    0     0     0     0     1     1     1     1     1
```

```
>> tf=~tf
```

```
tf =
```

```
    1     1     1     1     0     0     0     0     0
```

MATLAB 的这个特性是初学者应该仔细体会的。



## 2. 决策：控制流

决策控制流可以决定命令的执行流程，是计算机编程语言的重要功能。MATLAB 提供了三种决策或者控制流结构。它们是：for 循环、while 和 if-else-end 循环。它们的结构分别如下所示。

(1) for 循环。它的一般形式是：

```
for x=array
    命令语句
end
```

命令语句按数组（array）的每一列执行一次。

```
例， for n=1:10
        x(n)=sin(n*pi/10)
    end
```

执行结果为

```
x =
Columns 1 through 7
    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511    0.8090
Columns 8 through 10
    0.5878    0.3090    0.0000
```

(2) while 循环。其一般形式为：

```
while expression
    命令语句
end
```

只要在表达式（expression）里的所有元素为真，就执行命令语句。

(3) if-else-end 结构。一般形式为：

```
if expression
    命令语句
end
```

或者是

```
if expression
    命令语句
else
    命令语句
end
```

等等。

从结构上说，MATLAB 里的控制和决策流和其他的语言十分类似，即使是稍有变化，读者也可以从帮助里获得解答。但有一点是初学者必须注意的，在 MATLAB 里，许多在高级语言如 C 中，需要用控制流来实现的语句，在 MATLAB 里往往可以找到相应的数组替代方式。

如前面所举的关于 for 循环的例子，就可以用下面的命令替代。

```
>> n=1:10;  
>> x=sin(n*pi/10);
```

像这种情况下应该避免用控制流结构。

又如，实现将向量 x 中，大于 5 的元素的个数统计出来。首先读者最先想到的恐怕是 if 语句，代码可以这样写：

```
n=0;  
for i=1:length(x) %length(x)求向量 x 的长度  
    if x(i)>5  
        n=n+1;  
    end  
end
```

上面的代码，是从 C 等高级语言的编程思路得出的。但在 MATLAB 里，可以用更简洁的写法。

```
>> sum(x>5) % sum 函数将向量的元素求和
```

显然这种写法比前面的要简洁得多。

使用 for 循环和 while 循环的另外一个有用的技巧是，在语句执行之前，应预先分配数组。还是以前面讲的关于 for 语句的例子，在 for 循环内每执行一次命令，x 的尺寸就会加 1，这迫使 MATLAB 在每个循环花时间对 x 分配更多的内存，影响了执行的速度。为了消去这个步骤，这个 for 循环应重写为：

```
>> x=zeros(1,10); % 建立一个 10×1 的元素都为零的向量，起到了预先分配内存的作用  
>> for n=1:10  
    x(n)=sin(n*pi/10)  
end
```

当然，预先分配地址的思路在 C 里是很自然的，但由于 MATLAB 声明变量不需进行类型说明，也不需要预先分配内存，所以这一步读者在实际运用时往往会忘掉。

### 2.1.7 使用 MATLAB 在线帮助

至此，我想你应该对 MATLAB 程序语言设计的骨架已经有了比较深刻的认识，现在所

欠缺的只是一些血肉。比如说实现某种功能应该调用什么样的命令（函数）。为此，你可以阅读其他的 **MATLAB** 书籍，如果你对英文不恐惧，那可以直接看帮助文档，这里有最全面的信息。如果你知道某个函数的名字，但对其用法不熟悉，例如 `inv`，你可以

```
>> help inv
```

```
INV      Matrix inverse.
         INV(X) is the inverse of the square matrix X.
         A warning message is printed if X is badly scaled or
         nearly singular.
         See also SLASH, PINV, COND, CONDEST, NNLS, LSCOV.
```

这里要注意一点，一般的 **MATLAB** 函数名都是小写，帮助文档中函数名大写的目的是为了更方便阅读，在调用时切记要用小写。

但是，很多情况下，我们不知道实现什么功能该用什么函数，上面寻求帮助的方法，就不行了。一般情况下，**MATLAB** 函数的名字是有意义的，例如 `inv`，`det`，`who`，`ans` 等等，而且和通常使用的惯例是一致的，所以可以联想或试探，这样有些盲目性。比较好的方法是按主题查找。

```
>>help
```

```
HELP topics:
```

```
matlab\image           - (No table of contents file)
E:\matlab              - (No table of contents file)
VC98\Bin               - (No table of contents file)
cppmath\watcom         - (No table of contents file)
matlab\general         - General purpose commands.
matlab\ops             - Operators and special characters.
matlab\lang            - Programming language constructs.
matlab\elmat           - Elementary matrices and matrix manipulation.
matlab\elfun           - Elementary math functions.
matlab\specfun         - Specialized math functions.
matlab\matfun          - Matrix functions - numerical linear algebra.
matlab\datafun         - Data analysis and Fourier transforms.
matlab\polyfun         - Interpolation and polynomials.
```

```
..... (显示的内容太长，这里只选取了一部分)
```

```
For more help on directory/topic, type "help topic".
```

例如，你想知道求一个数的常用对数该调用什么函数，初步估计这个函数应该属于基本的数学函数，于是

```
>>help elfun
```

```

.....
Exponential.
exp          - Exponential.
log          - Natural logarithm.
log10        - Common (base 10) logarithm.
log2         - Base 2 logarithm and dissect floating point number.
pow2         - Base 2 power and scale floating point number.
sqrt         - Square root.
nextpow2     - Next higher power of 2.
.....

```

这样就可以知道是用 `log10` 这个函数。

有时候这种方法可能还不管用，因为它需要你确切知道要寻求帮助的主题。更好的方式是 `lookfor` 命令，它提供了按关键字搜索的能力，而关键词不必为函数名和命令名。其搜索范围是所有的 MATLAB help 标题，以及 MATLAB 搜索路径中 .m 文件的第一行，返回结果是包含指定关键字的那些项。例如：

```
>> lookfor complex
```

```

ctranspose.m: %'      Complex conjugate transpose.
COMPLEX              Construct complex data from real and imaginary parts.
CONJ                 Complex conjugate.
CPLXPAIR             Sort numbers into complex conjugate pairs.
IMAG                 Complex imaginary part.
REAL                 Complex real part
.....

```

这个命令的缺点是执行速度慢，有时候满足要求的项会很多，找起来不方便。

以上的命令都是基于命令行的，MATLAB 也提供了可读性好、易操作的图形帮助界面。你可以输入

```
>> helpdesk
```

或者用命令窗口的 Help 菜单下的 Helpdesk 命令来启动。这里的帮助文档是 HTML 格式的，MATLAB 还提供了 PDF 格式的许多电子书，其路径是“你的 MATLAB 所在的目录 \help\pdf\_doc”。

前面讲述的获得帮助的是 5.3 版本以前版本的帮助文档风格，对于使用 6.0 版本的读者，则可以不用这么麻烦。因为 6.0 版本的帮助界面则要友好得多。

图 2-3 显示了 MATLAB6.0 的帮助界面。在帮助窗体的左边，是帮助浏览器。它分成三个页面。在 content 页面，它按内容对帮助内容进行组织，可以让用户方便的浏览各个主题，包括 MATLAB 以及它的其他产品。对帮助内容，读者可以单击标题左边的加号将该内容展开。于是，读者不难发现，这些帮助主题是分层次组织好的，查询某个主题就显得很方便。而在其他的两个页面，则把帮助按索引进行分类，并提供了搜索功能。这就和 MSDN 很

类似了。

而右边的子窗口，则主要是显示帮助信息。读者可以按照自己的喜好来对帮助窗口的风格进行设计，具体的就是使用 View 菜单。

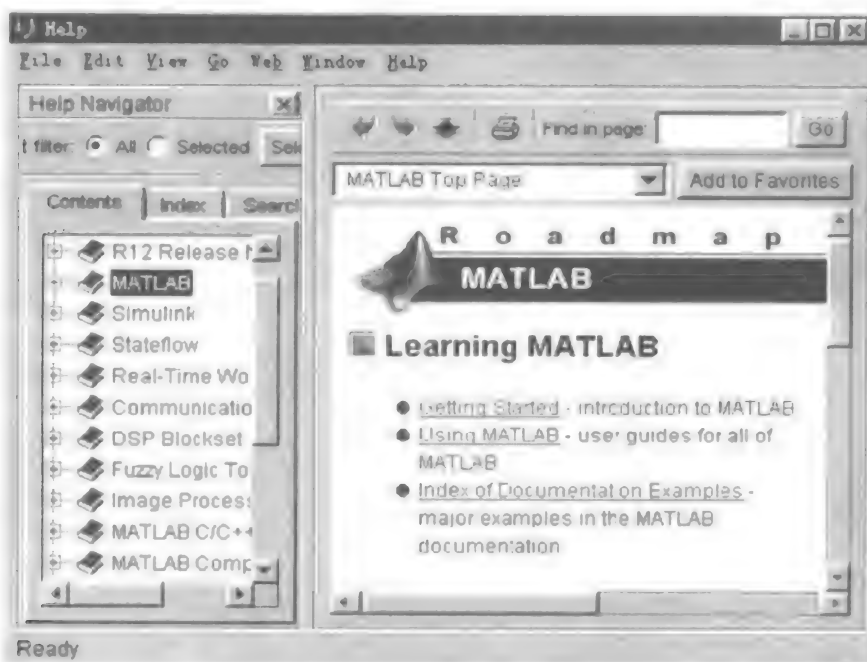


图 2-3 MATLAB 6.0 的帮助窗口

而前面提到的获得帮助的方法，MATLAB6.0 也是支持的，在有些场合使用命令行还是比较方便些。

为了方便读者，本书的附录 A 列出 MATLAB6.0 的一些常用函数。读者也可以购买其他的中文参考书，但最根本的方法还是学会看 MATLAB 的电子文档，毕竟它才是最全面和最权威的。

## 2.2 仿真应用：输入数据分析

在上一节对 MATLAB 的概要性叙述的基础上，这一节，将会着重介绍 MATLAB 里与仿真应用有关的一些函数。

这里介绍的顺序将按照第一章对仿真应用基本理论的框架顺序来安排，依次是随机变量的产生、数据的收集分析和仿真输出结果分析。而所讲述的内容是以 MATLAB5.3 以上版本为主的。

### 2.2.1 随机变量的产生

MATLAB 提供了两个基本的函数用于产生随机数。它们是 rand 和 randn。

#### 1. rand

它的作用是产生(0,1)间均匀分布分布的随机数或向量。其调用形式有

<code>Y = rand (n)</code>	返回一个 $n$ 维的 $U(0,1)$ 随机数方阵, $n$ 必须是一个标量
<code>Y = rand (m,n) 或 Y = rand ([m n])</code>	返回一个 $m \times n$ 维的 $U(0,1)$ 随机数矩阵, $n$ 必须是一个标量
<code>Y = rand (size(A))</code>	返回和矩阵 $A$ 相同维数的矩阵
<code>rand</code>	返回单个的随机数
<code>s = rand ('state')</code>	返回均匀分布随机数发生器的当前状态矢量, 其维数是 35。这种形式一般不常用。

MATLAB5 以上的版本使用了多种随机数发生器, 它可以产生 $[2^{-53}, 2^{53}]$ 内的所有浮点数。它的序列周期为  $2^{1492}$ 。

例

```
>> rand(3,4)
```

```
ans =
```

```
0.9528    0.5982    0.8368    0.3759
0.7041    0.8407    0.5187    0.8986
0.9539    0.4428    0.0222    0.4290
```

```
>>a=zeros(2,2)           % zeros 函数可以起到声明变量的作用
```

```
a =
```

```
0    0
0    0
```

```
>> rand (size(a))
```

```
ans =
```

```
0.1996    0.5383
0.3031    0.9102
```

```
>>(3-1)*rand (2,3)+1 % 产生[1,3]区间上的均匀分布随机数
```

```
ans =
```

```
2.0506    1.0689    2.5374
1.6137    2.4307    1.1190
```

## 2. randn

其作用是产生标准正态分布的随机数或随机矢量, 其调用形式和 `rand` 类似。

例

```
>>randn (1,2)

ans =

    -0.4326    -1.6656

>>randn (1,2) /2+1      %产生 N (1,0.25) 的正态分布。

ans =

    1.0627    1.1438
```

### 3. randperm

其作用是产生一个随机置换，调用形式

`p = randperm (n)` 产生 1:n 的一个随机置换。

例

```
>>p = randperm (4)

p =

     2     3     4     1
```

以 `rand` 和 `randn` 为基础运用我们在第一章中介绍过的方法可以产生给定分布的随机数，例如泊松分布和二项分布。`M` 文件函数 `posong` 演示了如何产生泊松分布的随机数。

```
function p=posong (a,n,m)
w=ones (n,m) * (-a) ;
th=exp (w) ;
r=ones (n,m) ;
dv=r>th;
p=zeros (n,m) ;
while nnz (dv) > 0
    r=r.* (rand (n,m) .*dv) ;
    dv=r>th;
    p=p+dv;
end
```

它的基本原理在第一章里已经作了介绍，这里我们对它的 MATLAB 的实现，简要地说明一下。参数  $a$  指泊松分布的参数， $n$ ,  $m$  则指返回的随机矢量的维数。这个程序中有些地方需要注意：

(1) MATLAB 支持矢量的逻辑运算，运算结果是一个元素为 0 或 1 的矢量。如 “`r>th`” 这条语句。

(2) 函数 `nnz` 的作用是统计一个矢量非 0 元素的个数。在这里的作用是判断是不是  $r$  中

所有的元素小于  $\exp(-a)$ 。

(3) 运算符 “ $\cdot$ ” 表示数组元素对元素的乘法，如  $[a_1, a_2, \dots, a_n] \cdot [b_1, b_2, \dots, b_n] = [a_1 \cdot b_1, a_2 \cdot b_2, \dots, a_n \cdot b_n]$

总之这个程序比较充分的利用了 MATLAB 基于向量的特点，仿照这个程序，不难写出产生二项分布随机数 M 文件函数，请读者自己尝试。

其实，像上面的这些工作读者大可不必去做，因为 MATLAB 的统计工具箱提供了一个各种分布随机数的发生器函数。为了使用这些函数，你必须安装 MATLAB 的统计工具箱。表 2-4 列出了这些函数。

表 2-4 统计工具箱里的随机数发生器

函 数 名	功 能 说 明
betarnd	Beta 分布
binornd	二进制分布
exprnd	指数分布
Frnd	F 分布
gamrn	伽马分布
geornd	几何分布
hygernd	超几何分布
nbinrnd	负二项分布
ncfrnd	非中心 F 分布
nctrnd	非中心 t 分布
normrnd	正态（高斯分布）
poissrnd	泊松分布
random	一指定的分布
raylrnd	瑞利分布
trnd	T 分布
unidrnd	离散均匀分布
unifrnd	均匀分布
weibrnd	韦伯分布

你可以通过 help 命令来获得这些函数的详尽使用方法。这里作为示例来看看 random 的用法。

4. random

它的作用是产生一个由 name 参数指定的分布的随机数。其调用形式为：

$R = \text{random}(\text{name}, A, M, N)$ ，name 指定分布的形式，A 说明该分布的参数，M,N 规定产生的随机数的矩阵维数，M 为行数，N 为列数，它们的缺省值都是 1（MATLAB 所有函数涉及维数的参数，顺序都是行在前，列在后）。由于 name 指定的分布可能不止一个参数，所以调用形式要做相应的变化。

$R = \text{random}(\text{name}, A, B, M, N)$ ，指定的分布有两个参数时使用。



**R=random (name, A, B, C, M, N)**, 指定的分布有三个参数时使用。

参数 **name** 的可以取以下这些值, 请详见 **MATLAB** 的帮助信息。

例, 泊松分布只有一个参数, 就要选用第一种形式来调用。

```
>>random ('poiss', 4, 2, 2)
```

```
ans =
```

```
6    4
6    1
```

### 2.2.2 输入数据的分析

在实际仿真中, 仿真模型的输入数据是要从实际中收集起来, 并要进行一定的分析。即便数据仅仅是像上一节一样由随机数发生器产生, 在使用数据前也要进行一番处理和检验。例如, **MATLAB** 中的随机数发生器产生的两列序列的相关性如何, 或者说是不是独立这都需要进行验证。在第一章中提到, 输入数据的处理程序是: 收集—>画出直方图—>参数估计—>拟合度检验。收集是需要人力去完成的, **MATLAB** 无法代劳, 但其使用极为方便的输入输出函数能使这些工作省力。而像后面的几个过程, **MATLAB** 都提供了相应的函数。

#### 1. 画直方图

**MATLAB** 中用于直方图绘制的命令有 **hist**、**histc** 和 **bar**。**hist** 的用途是根据输入的数据矢量画出直方图, 并返回每一小块的频数。区间划分的方式既可以等宽的, 也可以是不等宽的。调用形式为

**N=hist (Y)**, 其中 **Y** 指输入的数据向量, (这里既可以行向量, 也可以是列向量), 但如果 **Y** 为矩阵, 则看成几个列向量分别处理。由于没有指定区间如何划分, 函数缺省的将输入数据的范围等宽的划分为 10 块。输出 **N** 返回一频数矢量。

**N=hist (Y, M)**, 其中 **M** 是一个标量, 指定分块的个数。

**N=hist (Y, X)**, 其中 **X** 是一个向量, 用以指定每块的中心值, 这就使得用户具有定义非等宽分块的能力, 如果想更随意的定义块的边界就要使用 **histc** 函数。此外, **hist** 还可以返回每个小块的中心值, 其调用形式为 **[N, X]=hist (...)**, 如果调用时没有输出参量, 则函数仅画出直方图。

例

```
>>y=random ('poiss',10,500,1); %产生 500 个泊松分布的随机数据样本
```

```
>>n=hist (y)
```

```
n =
```

```
5    35    65   127   123   46    53   29   13    4
```

图 2-4 是执行输出的直方图。10 个小区间一般不利于你对数据的分布形式作出判断, 可以增加区间的个数

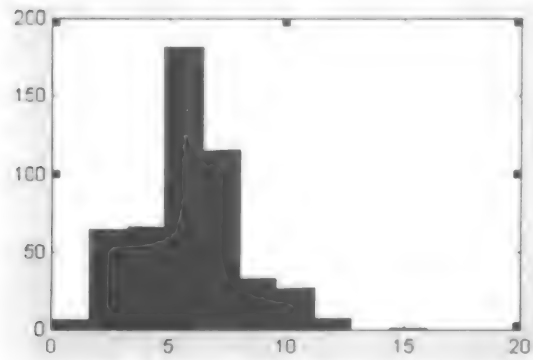


图 2-4 泊松分布直方图 (10 个分块)

```
>>n=hist (y,25); %区间个数变为 25
```

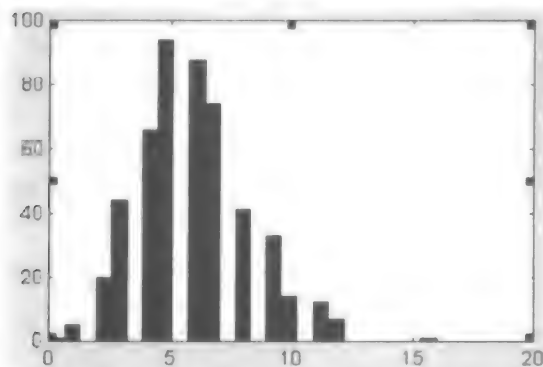


图 2-5 区间数为 20 的泊松分布直方图

也可以自定义区间划分方案。

```
>>y=randn (5000,1); %生成 5000 点的正态分布随机数
>>x=linspace (-2.5, 2.5, 15); %生成小区中点向量 x
>>hist (y, x);
```

其中，函数 `linspace (first, last, n)` 用来创建从 `first` 开始，到 `last` 结束，元素个数为 `n` 的行向量。注意，`hist (Y, X)` 并不要求说明小区间终点的向量一定是列向量，所以上面的用法是没有问题的。图 2-6 是它的输出结果。

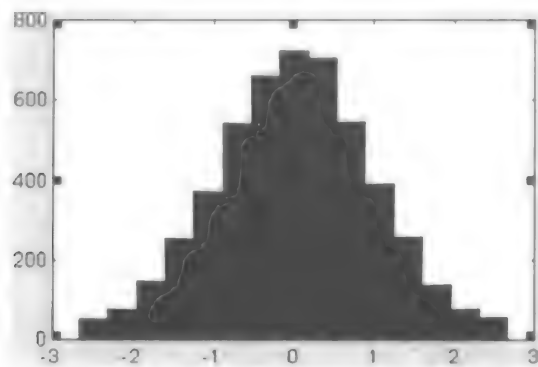


图 2-6 正态分布的直方图

比较高斯分布和泊松分布的直方图，不难发现它们之间形状上的差别，高斯分布的直方图一般关于某条轴对称，而泊松分布稍微向左边偏一点。这些特征分别与各自的概率密度函数图像相对应。至此就可以对输入数据的分布形式作出判断，再进行参数估计。

## 2. 参数估计

对一个组观测数据进行参数估计，一般都要先计算样本均值和样本方差。MATLAB 中计算这两个统计量的函数分别是 `mean` 和 `std`。

`mean` 函数的调用形式为 `m=mean (X)`，当 `X` 为一个向量时，函数返回各元素的平均值；如果 `X` 是一个矩阵，则返回一个行向量，其元素对应矩阵每列的平均值。

例

```
>> d = rand (10,3)
```

```
d =
```

0.7344	0.1841	0.7875
0.4894	0.9840	0.4358
0.4445	0.3412	0.5083
0.2214	0.2123	0.9805
0.1488	0.9780	0.5550
0.1927	0.3345	0.0823
0.3017	0.6318	0.6326
0.8630	0.8767	0.8420
0.1011	0.2020	0.3715
0.7868	0.6137	0.2647

```
>> mean (d )
```

```
ans =
```

0.4284	0.5358	0.5460
--------	--------	--------

是不是只能按列来求就平均呢？当然不是。你也可以使用下面的形式。

`m=mean (X, DIM)`，作用是按 `DIM` 维求平均。

```
>> mean (d, 2)
```

```
ans =
```

0.5687
0.6364
0.4313
0.4714
0.5606
0.2032
0.5220

```

0.8605
0.2248
0.5551

```

一定要注意维数的含义，不能把 DIM 看成指第 DIM 行或 DIM 列。上例中，矩阵的维数为 2，其每个元素可以用下标 (a, b) 来表示，mean (d, 2) 表示求均值时第 2 个下标在变，即对每行求平均。

求矩阵 d 的标准方差

```

>>std (d)
ans =
    0.2818    0.3242    0.2750

```

它的用法和 mean 函数很类似。但在使用时一定要分清出标准方差和样本的二阶矩的区别。在统计里，标准方差的归一化系数是  $1/(N-1)$ ，而二阶矩的归一化系数则是  $1/N$ ，这里的  $N$  是样本序列的长度。对这个区别不明白的读者，可以查阅有关书籍。Std 提供了一个 flag 标志选用何种计算模式。调用形式为 std (X, [flag], [DIM])， “[ ]” 表示此参数是可选参数。flag 大于 1 表示求二阶矩，为 0 表示求标准方差。DIM 参数的意义和 mean 函数一样。

这种有多个输入参数的 MATLAB 函数，调用时要注意参数的写法。例如，std (d, 0, 2)，std (d, 2) 等都是正确的，但 std (d, 2) 这种形式是错误的。

而且还要注意的，std (...) 返回的值是平方根后的值。

计算出这两个统计量，就可以估计大部分随机分布的参数了。例如，泊松分布的  $\mu$  的估计量就等于样本均值，指数分布参数  $\lambda$  的估计量为样本均值的倒数，正态分布的  $\mu$  和  $\sigma^2$  的估计量分别是样本均值和标准方差（关于各种分布的参数估计量详见第一章）。

有些分布的参数估计还要用到别的统计量。例如 (0, b) 上的均匀分布，b 的估计量就要求样本点最大值。这一点在 MATLAB 中也是很容易计算的。

max 函数和 min 函数分别用来求最大值和最小值，其用法同 mean。

```

>> max (d) , min (d) %分别求矩阵 d 每列的最大值和最小值。

```

```

ans =
    0.8630    0.9840    0.9805
ans =
    0.1011    0.1841    0.0823

```

于是该均匀分布的参数估计

```

>> (10/9)*max (d)
ans =
    0.9589    1.0933    1.0894

```

在 MATLAB 里还有许多的函数用于数据分析，这些函数缺省情况都是按列向量进行的，但都可以像前面 mean 函数的方法自定义成按行向量进行分析。表 2-5 列出了这些函数。

表 2-5 数据分析函数

函 数 名	参 数 列 表	功 能 说 明
Corrcoef	(X), (X,Y)	求相关系数
Cov	(X), (X,Y), (X,Y,flag), (X,flag)	协方差矩阵
Cplxpair	(X), (X,TOL)	把向量分类成复共轭对
Cross	(X,Y), (X,Y,DIM)	向量的向量积（叉积）
Cumprod	(X), (X,DIM)	向量的累计积
Cumsum	同 cumprod	向量的累计和
del2	(U,HX,HY),	5 点离散拉氏算子
Diff	(X), (X,N), X (X,N,DIM) (N 表示 N 阶矩)	计算元素之间差
Dot	Y=dot (X,Y)	向量的点积（内积）
gradient	[FX,FY]= gradient (F,HX,HY)	近似梯度
median	Y=median (X), Y=median (X,DIM)	向量的中值
prod	同 median	向量的积
Sort	[Y, I]=sort (X), (I 的下标序列)	按升序排列
sum	同 median	向量的元素和

这里以 sort 函数为例。

```
>>d = 4 : -1 : 1
```

```
d =
```

```
4     3     2     1
```

```
>>[Y, I]= sort (d )
```

```
Y =
```

```
1     2     3     4
```

```
I =
```

```
4     3     2     1
```

这里，I 就是向量 Y 按元素值升序排列后，下标的相应排列。

按照前面的讲述，进行参数估计的步骤是首先计算输入数据的样本均值，标准方差或者其他的统计量，再按照分布的参数估计式进行计算。

MATLAB 可以将这个过程缩短，在其统计工具箱里，有许多函数专门用来进行参数估计。

例如，估计泊松分布的参数，可以用 poissfit 函数，它是 toolbox \stats 统计工具箱中用于

参数估计的众多函数之一。其他的函数 `betafit`、`binofit`、`expfit`、`gamfit`、`mle`、`normfit`、`unifit` 和 `weibfit`。其中 `mle` 是指极大似然估计，十分重要的一种估计方法。MATLAB 工具箱里的参数估计函数都提供了两种估计形式，一种称为点估计，另外一种则是区间估计。以函数 `poissfit` 为例，它有两种调用形式：

`A=POISSFIT (X)` 及

`[C, BIN] = POISSFIT (X, ALPHA)`

第一种形式返回的结果就是泊松分布参数  $a$  的点估计值，也就是我们在第一章里介绍过的。第二种结果则返回一个置信概率为  $100(1-ALPHA)\%$  的区间估计，输出参数中 `C` 返回区间的中点值，`BIN` 返回区间的左右端点值。`ALPHA` 的取值一般为  $0.05$ ，也就是置信概率为  $95\%$ 。

例

```
>> x = random ( 'poiss', 6, 25, 1 ) ; %
```

```
>> a = poissfit ( x )
```

```
a =
```

```
6.4400
```

```
>>[a , b ] = poissfit ( x , 0.05 )
```

```
a =
```

```
6.4400
```

```
b =
```

```
5.4452
```

```
7.4348
```

其他的参数估计函数，除了参数个数有所不同外，用法与 `poissfit` 基本上差不多。但极大似然估计函数 `mle` 稍有不同，它也具有点估计和区间估计两种形式，但要传一个参数 `'DIST'` 以指明对何种分布进行估计。

```
>> a= mle ( 'poiss', x , 0.05 )
```

```
a =
```

```
6.4400
```

### 3. 拟合度检验（假设检验）

拟合度检验是输入数据处理的第三步，也是最关键的一步，对仿真的效果有很重要的影响。按照第一章讲述的伽马检验理论，拟合度检验要分作以下几个步骤：

(1) 将样本值（观测数据）分组；

(2) 计算每一组的观测频数，以及相应的理论频数，根据公式计算出检验的统计量；

(3) 根据样本的自由度，以及检验前给定的显著性水平，在卡爱平方表里查出作出判决的阈值，并将检验统计量与之相比较。若大于阈值，则说明先前假设的分布和观测数据拟合得不好，要重新作出假设；若小于，则说明把观测数据看成此分布比较合理。

MATLAB 里, 前面介绍的函数 `hist` 可以完成将观测数据分组以及计算观测频数, 除此之外, 函数专门 `histc` 用来完成这个功能。`Hist` 最基本的调用形式为: `N=HISTC (X, EDGES)`, 它的作用是计算向量 `X` 中的数据落在向量 `EDGES` 相邻元素间的个数, 返回值是一个向量。请注意, `EDGES` 必须是个单调非减向量, 即后面元素的值不能小于前面的元素, 否则会报错。向量 `N` 里第 `k` 个元素 `N (k)`, 表示满足  $\text{EDGES} (k) \leq X (i) < \text{EDGES} (k+1)$  这一条件, 所有 `i` 的个数。亦即观测数据 `X` 落在  $[\text{EDGES} (k), \text{EDGES} (k+1)]$  这一区间的频数。

```
>> x = random ('poiss', 6, 500, 1) ;
>> e = 1 : 8 ;
>> n = histc (x, e)
```

```
n =
```

```
5
20
44
68
80
72
73
51
```

也可以让 `hist` 返回 `X` 中没一个数据所落入区间的索引, 使用方法是 `[N, BIN]= histc (X, EDGES )`。

理论频数的计算是实现拟合度检验的一个难点, 实现起来要稍费周折。在 MATLAB 里没有一个函数能直接实现这一点, 需要一点小小的技巧。

在 MATLAB 统计工具箱里有两类函数涉及到计算分布的概率。一类函数通常用来计算某一分布在某个值上的概率密度值, 这一类函数的名称一般由分布的简称加后缀 `pdf` (详情请查阅 MATLAB 帮助文档中关于统计工具箱的内容)。另外一类则是计算累计分布函数值, 其函数名以后缀 `cdf` 结尾。这里以正态分布为例来说明这一类函数的用法。

与正态对应的函数为 `normcdf`。其调用形式为 `P= NORMCDF (X, MU, SIGMA)`, `P` 返回正态分布在向量 `X` 的数据点上的的累计分布函数值。其中 `MU` 和 `SIGMA` 是正态分布的参数, 分别对应均值和标准方差, 它们的缺省值分别是 0 和 1。

例如

```
>> x=[ 0.4  0.1  0.2  0.3  1  1.5 ]; %定义输入向量 x
>> normcdf (x, 0, 1) %X 对应的正态分布累计函数值
```

```
ans =
```

```
0.6554    0.5398    0.5793    0.6179    0.8413    0.9332
```

概率论里有一个很基本的关系

$$P([a,b]) = P((-\infty,b]) - P((-\infty,a]) \\ = cdf(b) - cdf(a)$$

而如上所示, MATLAB 具备计算一个累计分布函数值的能力, 于是也就能计算一个区间的概率值, 从而算出分组的理论频数。

例如, 现在要计算 [0.1 0.2 0.3 0.4 0.5 0.6 0.7] 这个分组的标准正态分布的理论频数。计算顺序如下

```
>> b= normcdf (x, 0, 1 )

b =
Columns 1 through 7
0.5398    0.5793    0.6179    0.6554    0.6915    0.7257    0.7580

Columns 8 through 10
0.7881    0.8159    0.8413

>> diff (b)

ns =
Columns 1 through 7
0.0394    0.0387    0.0375    0.0360    0.0343    0.0323    0.0301

Columns 8 through 9
0.0278    0.0254
```

可以把以上过程写成一个 M 函数。

## 2.3 仿真应用之输出分析

### 2.3.1 图形函数

将仿真结果可视化, 是输出分析的常用的一种方法。MATLAB 语言提供了一套功能强大的图形程序, 是实现这一过程的极佳手段, 这里介绍的只是一些较常用的图形函数。

#### 1. 函数: figure()

**h=figure** 创建一个图形窗口并返回它的句柄;

**figure (h)** 使具有句柄 h 的窗口成为当前窗口, 其效果是位于最顶层处于可视状态。

如果窗口 H 不存在, 则将创建一个句柄为 h 的图形窗口。

#### 2. 画图函数: plot()

**plot (x, y)** 绘制 y 对于 x (即以 x 为横轴) 的图形。则矩阵的行向量和列向量将被绘



制。

`plot (y)` 绘制对 `y` 索引（数据点数）的 `y` 图形。

`plot (x, y, s)` 用字符串 `s` 指定的不同类型、样式和颜色的线条，来绘制图形。表 2-6 列出了其具体的定义。

在上述调用形式中，如果 `y` 是复数域的向量，则分别对它的实部和虚部绘图，如图 2-7、图 2-8 所示。

表 2-6 基本线型和颜色

符 号	颜 色	符 号	线 形
y	黄色	.	点
m	紫红	o	圆圈
c	青色	x	x-标记
r	红色	+	加号
g	绿色	—	实线
b	蓝色	:	点线
w	白色	-.	点划线
k	黑色	--	虚线
		*	星号

例如：

```
>> x = linspace (0, 2*pi, 30) ;
>> y= sin (x) ;
>> plot (x, y) ;
>> plot (y, x) ;
```

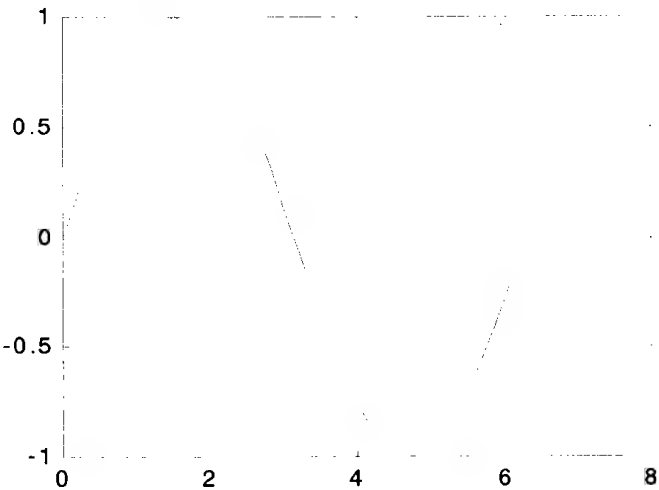


图 2-7 绘图命令 plot 示例

上面蕴涵了一个如何将一个图旋转 90° 的技巧。

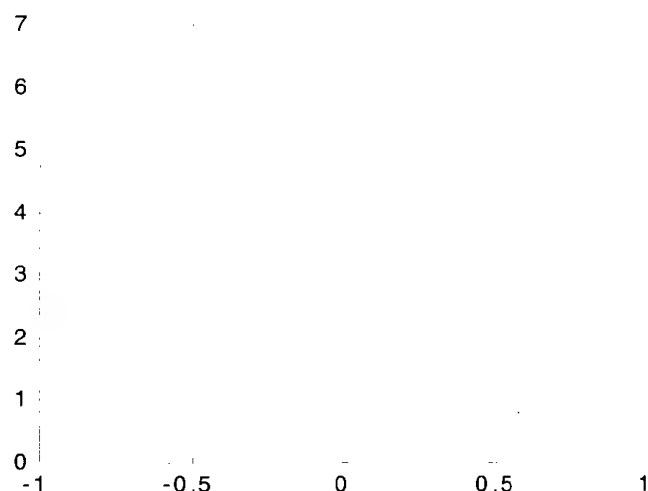


图 2-8 将图形旋转 90°

下面是使用不同线形，颜色和标记点的例子，如图 2-9 所示。

```
>> z = cos (x) ;
>> plot (x, y, 'b:', x, z, 'r-', x, y, 'ko', x, z, 'c+' ) ;
```

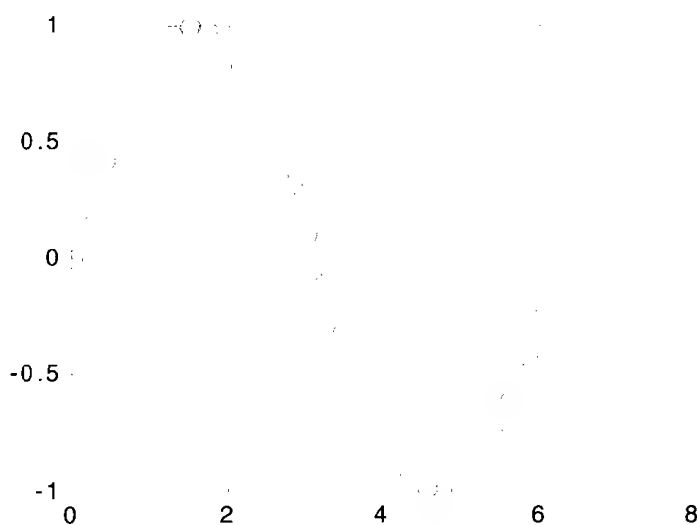


图 2-9 使用不同线形绘图

MATLAB 还提供了许多的配套函数，使用这些函数，用户可以定制绘图的效果。例如，可以在当前图像加栅格、为图形添加标题、标记横坐标轴和纵坐标轴、加图例、以及定制图形坐标轴等等，如图 2-10 所示。

```
>> x = linspace (0, 2*pi, 30) ;
>> y=sinx
>>plot (x,y,x,z,'r:')
>>xlabel ('横坐标 X')
```

```
>>ylabel('纵坐标 Y、Z')
>>title('正弦和曲线图')
>>legend('sin(x) ','cos(x) ')
```

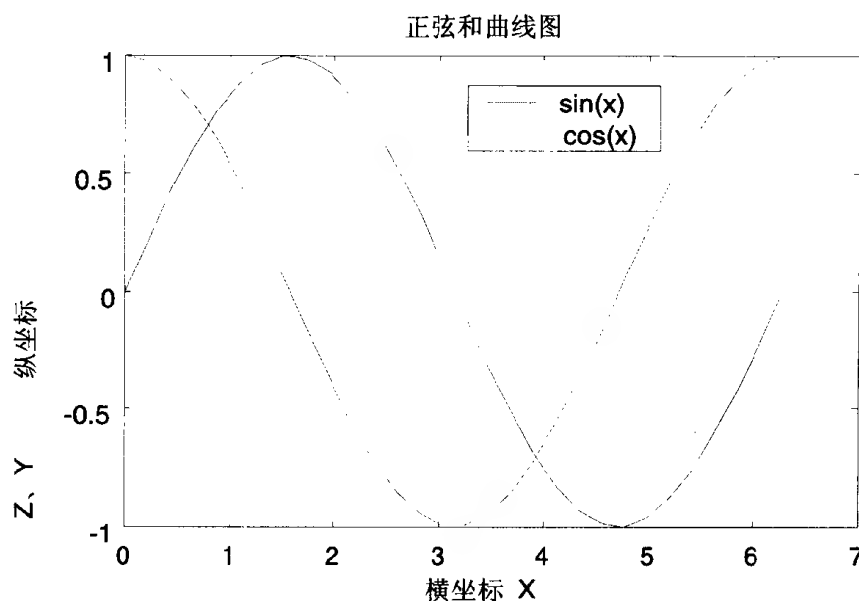


图 2-10 定制绘图效果

从前面的例子看到，`plot(x1,y1,x2,y2…)` 这种调用方法可以在一个图形上绘制多条曲线。那又如何能在已存在的图形中添加新的曲线呢？是不是调用发出新的 `plot` 命令就可以了？答案是否定的。一般情况下，如果在一个 `plot` 命令发出后，再调用新的 `plot` 命令，MATLAB 会取消原先绘制的曲线，这样就无法达到添加的目的。这时就要设置图形的保持属性，即保持当前图形与所有的坐标特性以便在已存在的图形上再添加其他图形。这个属性由 `hold` 来控制，与之相关的命令有：

- (1) `hold on`            置当前图形的 `hold` 属性为 `on`。
- (2) `hold off`           置当前图形的 `hold` 属性为 `off`。
- (3) `ishold`            返回 `hold` 属性的值。

还是沿用前面的正弦和余弦例子来说明它们的使用

```
>>plot(x,y) % x,y,z,均使用前面例子中的定义
>> hold on
>> ishold

ans =

     1

>> plot(x,z,'m*:')
>> hold off
>> ishold
```

```
ans =
    0
```

绘出的结果如图 2-11 所示。

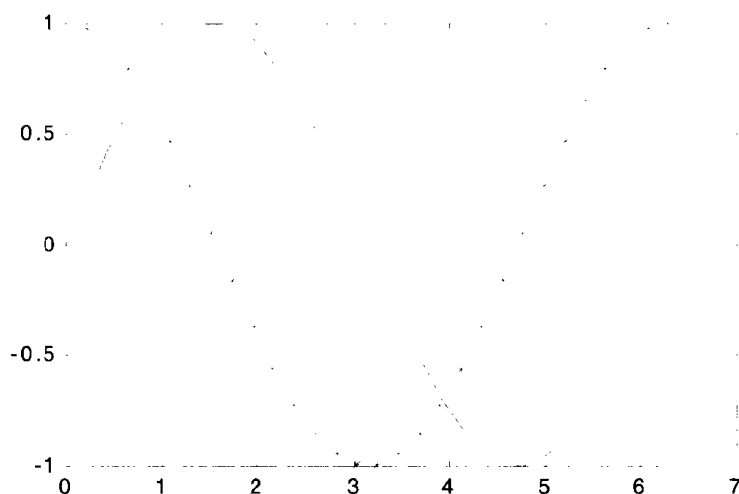


图 2-11 使用 hold on 命令示例

### 3. 子图绘制: subplot()

`subplot(m, n, p)` 将窗口划为  $m \times n$  的小窗口阵列, 选择  $p$  个为当前绘图窗口并返回句柄。窗口的计数顺序为从顶部的行开始, 然后再到下一行。

下面的例子说明了 `subplot` 的用法, 因为命令较长, 所以写成了 M 文件的形式。

```
x=linspace (0, 2*pi, 30);
titles = ['sin (x) ','cos (x) ','sin (2x) ','tan (x) '];
y= zeros (4, 30);
y (1,:) = sin (x);
y (2,:) = cos (x);
y (3,:) = 2* sin (x) .* cos (x);
y (4,:) = sin (x) ./ (cos (x) + eps);
for i=1:4
    subplot (2, 2, i);
    plot (x, y (i,:));
    title (titles (i,:));
    axis ([0 2*pi -1 1]);
end;
```

图 2-12 显示了命令执行完后的绘图结果。

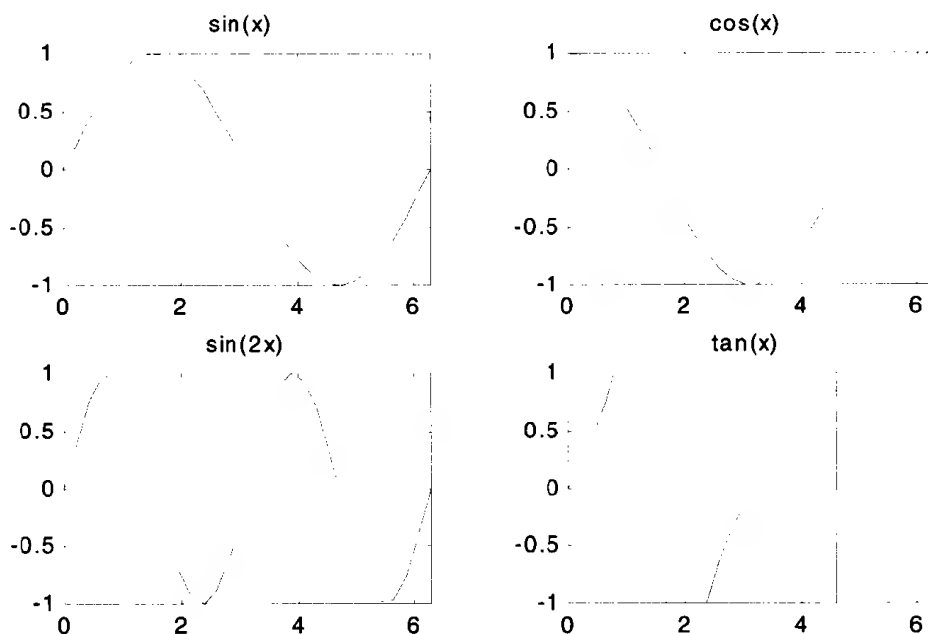


图 2-12 子图绘制结果

#### 4. ginput()

在某些情况下，读者希望在当前窗口选择坐标点。在 MATLAB 中，`ginput` 命令实现了这项功能。其调用形式为

`ginput` 用鼠标或光标获得坐标输入。

`[X, Y]=GINPUT(N)` 从当前坐标轴获取  $n$  个点及相应的  $X, Y$  坐标，这些值存放在长度为  $n$  的  $X, Y$  坐标向量中。其中的光标可以用鼠标定位。按鼠标键或键盘上的任意键输入数据。回车可提前终止输入。如果不说明  $N$  的值，则表示采集无限个数直至按回车键。

`[X, Y, BUTTON]=GINPUT(N)` 返回的第三个结果 `BUTTON`，它包含一个整数型向量，指明鼠标用过哪个键（从左起 1, 2, 3），如果使用过键盘的键，则返回相应的 ASCII 码值。

### 2.3.2 曲线拟合与插值

在进行仿真结果的分析时，人们经常面临用一个解析函数来描述仿真结果与某些参数的关系的任务。对这个问题有插值和拟合两种方法。在插值法里，数值假定是正确的，要求以某种方法描述数据点之间所发生的情况。而拟合方法的目的是寻找一条光滑曲线，它在某种准则下最佳地拟合数据。

前面一节中所讲的 `plot` 函数中连接数据点的线条，实际就是用插值得到的。

#### 1. 曲线拟合

曲线拟合涉及回答两个基本问题：最佳拟合最佳是什么准则下的最佳，应该用什么样的曲线。常用的拟合方法是多项式的最小二乘拟合，其准则是最小误差平方和准则，所用的曲线限定为多项式。

在 MATLAB 里，函数 `polyfit` 用于求解最小二乘曲线拟合问题。其说明如下

`polyfit` 最小二乘拟合，调用形式

`polyfit(x, y, n)` 找到对输入数据  $y$  的  $n$  阶最小二乘多项拟合多项式  $p(x)$  的系数。

例,

```
>>x=linspace (0,1,11) ;
>>y=[-4.47 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2];
>>p=polyfit (x,y,2)
```

p =

```
-9.8108    20.1293   -0.0317
```

```
>> z= polyval (p,x) ;
>>p2= polyfit (x,y,10)
>> plot (x,y,'-om',x,z,'-*r')
```

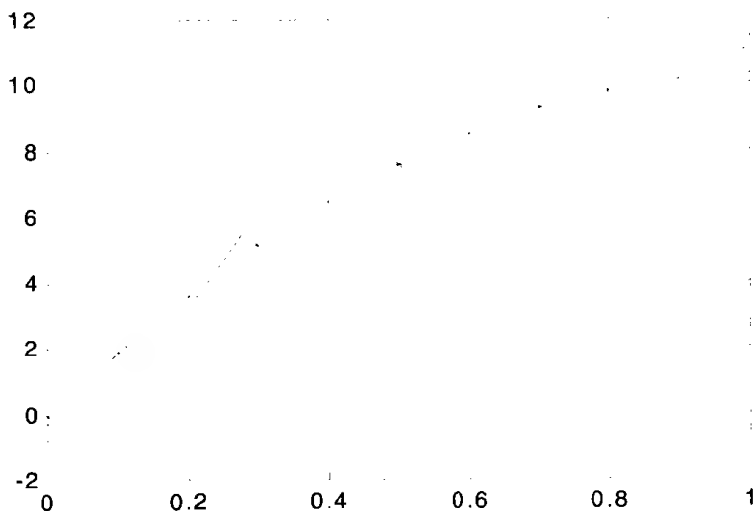


图 2-13 拟合和 plot 命令

图 2-13 是命令执行得到的结果。关于多项式的阶数并不是越多越好。因为在进行高阶曲线拟合时，经常会产生纹波现象。

## 2. 一维插值

插值定义为对数据点之间的函数的估值方法，这些数据点是由某些集合给定。当人们不能很快求出所需中间点的函数值时，插值就是一种有价值的方法。

其实，前面讲过的 plot 命令或许就是最简单的插值例子。按照缺省，MATLAB 用连线连接所有的数据点来作图。这种插值也称为线性插值。它的效果随数据点个数的增加和它们之间距离的减小时，而变得更好，线性插值更精确。按照自变量的个数可以对插值进行归类，如果只有一个自变量，就称为一维插值。

在 MATLAB 里，函数 interp1 用于一维插值。它的用法描述如下。

**YI = INTERP1 (X,Y,XI)** 用插值方法来寻找 XI 向量对应的函数值，该函数由输入向量 X 和 Y 来指定，其中 Y 是函数在自变量为 X 对应的函数值。当 Y 是一个矩阵时，插值就按 Y 的每一列来进行。如果传入的参量没有指定自变量 X，那么就假定 X=1:N，这里 N 是 Y 的

长度。

例如，

```
>> x = 0:10; y = sin (x) ; xi = 0:.25:10;
>> yi = interp1 (x, y, xi) ;
>> plot (x, y, 'o', xi, yi)
```

在上面，根据已知的数据点  $x$  和  $y$ ，来估计  $x_1$  的函数值。图 2-14 是将插值结果绘出的结果。

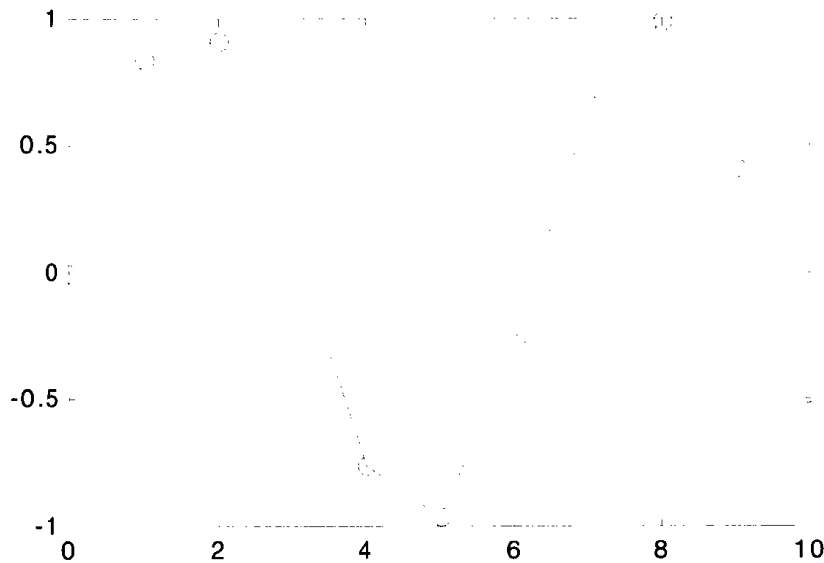


图 2-14 根据插值结果绘出的图

缺省情况下，`interp1` 函数采用线性插值。但它也允许使用者选择所使用的插值思想。调用形式为 `YI = INTERP1 (X,Y,XI,'method')`。所支持的思想有：

- (1) 'nearest' ——最近临域插值
- (2) 'linear' ——线性插值
- (3) 'spline' ——三次样条插值
- (4) 'cubic' ——三次插值

所有的插值思想都要求  $X$  必须是单调递增的。 $X$  可以是非均匀划分的。当  $X$  是等距的和单调递增的，可以使用思想：'\*linear'，'\*cubic'，'\*nearest'。在  $X$  为非均匀分布时，进行更快速的线性插值，可以使用 `interp1q`。

例如，

```
>> yi = interp1 (x, y, xi,'spline') ;
>> plot (x,y,'o',xi,yi)
```

在上例中，用三次样条进行插值，图 2-15 是根据插值结果绘出的图形，从图中可以看出，绘出的结果比线性插值的结果要光滑得多。

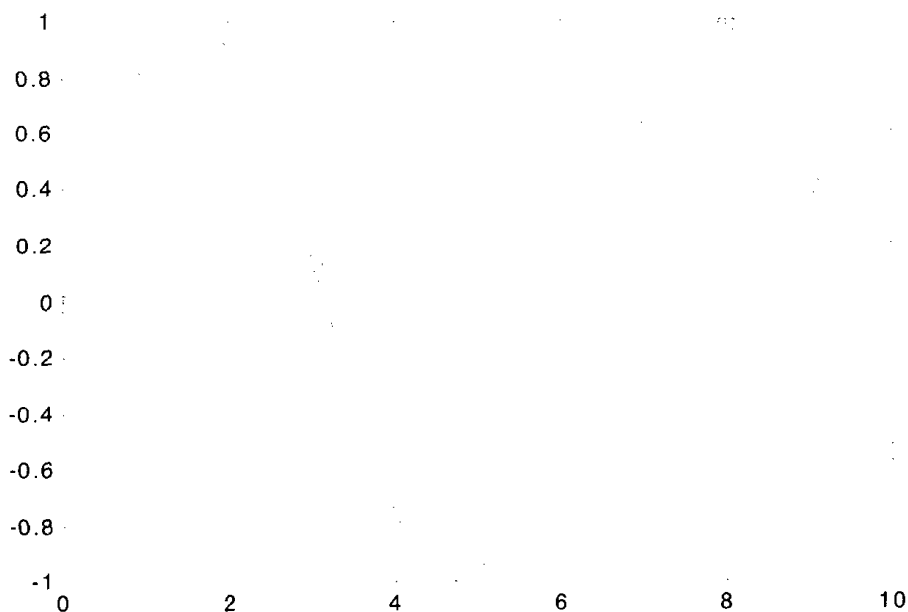


图 2-15 三次样条插值的结果

### 3. 三次样条插值

学过数值计算的读者知道，使用高阶多项式的插值常常产生病态的结果。消除病态的方法又很多种，三次样条是其中最常用的一种。在 MATLAB 中，实现基本的三次样条插值的函数有 `spline`、`ppval`、`mkpp` 和 `unmkpp`。例如，

```
>> x = 0:10; y = sin(x);
>> xx = 0:0.25:10;
>> yy = spline(x,y,xx);
>> plot(x,y,'o',xx,yy)
```

在三次样条中，要寻找三次多项式，以逼近每对数据点间的曲线。在最简单的用法中，`spline` 获取数据  $x$  和以及期望值  $xi$ ，寻找拟合  $x$  和  $y$  的三次样条内插多项式，然后，计算这些多项式，对每个  $xi$  的值，寻找相应的  $yi$ 。将插值得到的结果，用 `plot` 绘出的图如图 2-16 所示。

```
>> circle = spline(0:4, [0 1 0 -1 0 1 0; pi/2 0 1 0 -1 0 pi/2]);
>> xx = 0:0.1:4;
>> cc = ppval(circle, xx);
>> plot(cc(1,:), cc(2,:))
```

在调用 `spline` 时，如果只有前面的两个参数，返回的结果是三次样条多项式系数或者分



罗仿真，希望读者能仔细体会这种方法。这两个例子，还有一个共同点就是都是静态仿真，静态仿真是相对于动态系统仿真而言的，也就是说仿真实现不需模拟时钟的推进。

这里举例的目的不是为了演示运用 MATLAB 编程的技巧，而是为了让读者对仿真有一个大致的概念。

### 2.4.1 二进制通信系统的蒙特卡罗仿真

#### 1. 问题

通信理论时常要面对的一类问题是，估算某种通信方式的误码率。比如说不同的调制方式和不同信道环境对误码率的影响。尽管解析的方式可以推导出一些情况下的结果，但更多的情况下解析方法无能为力。这时候，计算机仿真是一个很有效的手段。这个例子要涉及的情况是其中最简单的一种，但在对方法本质的体现上是没有分别的。

#### 2. 模型的建立

在二进制通信系统中，由 0 和 1 组成的二进制数据采用两个波形来传输，假设数据率为  $R\text{bit/s}$ ，发送每个比特都根据如下规则映射为相应的波形

$$0 \rightarrow s_0(t), 0 \leq t \leq T_b$$

$$1 \rightarrow s_1(t), 0 \leq t \leq T_b$$

式中， $T_b$  是时间间隔。这个映射的过程也称为脉冲调制。在通信理论里，通常假定数据比特中 0 和 1 出现的概率相同，各为二分之一。

而对于传输信号的信道，通常用是加性高斯白噪声信道（AGWN）来近似。也就是在传输的信号上叠加一个高斯随机过程。因此得到接收波形为

$$r(t) = s_i(t) + n(t), i = 0, 1, 0 \leq t \leq T_b$$

在接收的一端，要根据在间隔  $0 \leq t \leq T_b$  内收到的信号  $r(t)$  来判断接收到了 0 还是 1。根据尽可能多减少差错率的原则所设计出的接收机称为最佳接收机。

在高斯加性白噪声信道的假设下，相应的最佳接收机是由信号相关器或匹配滤波器、判决器两个模块组成。其结构如图 2-18 所示。

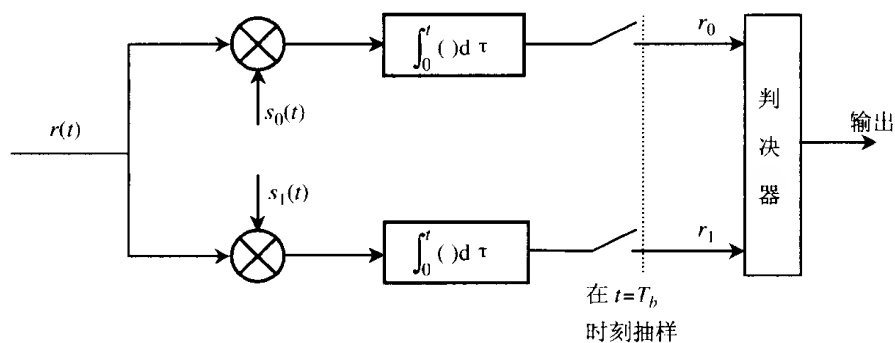


图 2-18 AGWN 信道下的最佳接收机框图

选择不同传输信号，相关器的输出就会有不同结果。例如，如果传输信号  $s_0(t)$ 、 $s_1(t)$  满

足下面的条件

$$\int_0^{T_b} s_0(t)s_1(t)dt = 0$$

$$\int_0^{T_b} s_0^2(t)dt = \int_0^{T_b} s_1^2(t)dt = E$$

第一个等式表述的意思是指两个信号正交，后一个等式要求两个信号的能量相同。在这种情况下可以导出

在 0 被发送的前提下

$$r_0(t) = n_0 + E$$

$$r_1(t) = n_1$$

相应的，在 1 被发送的前提下

$$r_0(t) = n_0$$

$$r_1(t) = n_1 + E$$

可以证明，当  $n(t)$  是功率谱为  $N_0/2$  的高斯白过程时， $n_0$  和  $n_1$  都是均值为零，方差为  $EN_0/2$  的高斯随机变量，且互为独立。

相关器的判决准则是当  $r_0 > r_1$  时就判定接收波形代表的比特是 0，否则就判为 1。在这种准则下，有可能会造成当实际发送 0，却判为 1 的错误，这就是误码了。

在这里，误码率是可以解析的方式推导出来，其结果是

$$P_e = Q(\sqrt{E/N_0})$$

在通信理论中，通常定义信噪比这个参数来作为系统的一个参数，它的定义为

$$SNR = 10\log_{10}(E/N_0)$$

下面将演示如何用 MATLAB 实现计算机仿真来确定误码率和信噪比的关系，当然对于前面的推导情况只要做一个简单推导就可以得出误码率和信噪比的关系。

### 3. 二进制通信系统的蒙特卡罗仿真

根据前面导出的简化模型，不难画出仿真的系统模型。正如图 2-19 所显示的那样，整个系统可以分为四个模块：二进制数据源产生模块，信道干扰模块，判决器模块，最后为了统计误码率还要一个比较计数模块。二进制数据源表示要发送的数字信号，根据前面的定义它应该 0 和 1 组成的等概率的前后独立二进制序列（这里指不同时刻的抽样变量独立）。其产生过程为：先使用随机数发生器产生一个 (0,1) 均匀分布的随机数，如果产生的数在 [0, 0.5] 范围内，二进制信源的输出为 0；否则输出为 1。信道干扰模块的主要作用是根据数据源的输出是 0 还是 1 来确定判决变量  $r_0$  和  $r_1$  的输出。为了模拟噪声，要产生两个相互独立的高斯分布的随机数序列，其均值为 0。整个模块的效果是：如果数据源的输出为 0 则  $r_0=n_0+E$ ， $r_1=n_1$ ；如果数据源输出为 1，那么  $r_0=n_0$ ， $r_1=n_1+E$ 。判决器的原理是比较  $r_0$  和  $r_1$ ，如果  $r_0 > r_1$  则判决

器的输出 0, 否则为 1。比较计数模块的功能是将判决器的输出和数据源的原始序列进行比较, 并统计它们不相同的次数, 得出误码率。

仿真的目的是画出误码率和信噪比的关系曲线, 并最终和理论计算的结果比较。为此要在仿真中要不断的改变信噪比  $SNR = 10\log_{10}(E/N_0)$ , 为了方便起见, 规定 E 的值始终为 1, 于是相应的两个高斯随机变量的方差为  $0.5e^{(-SNR/10)}$ 。

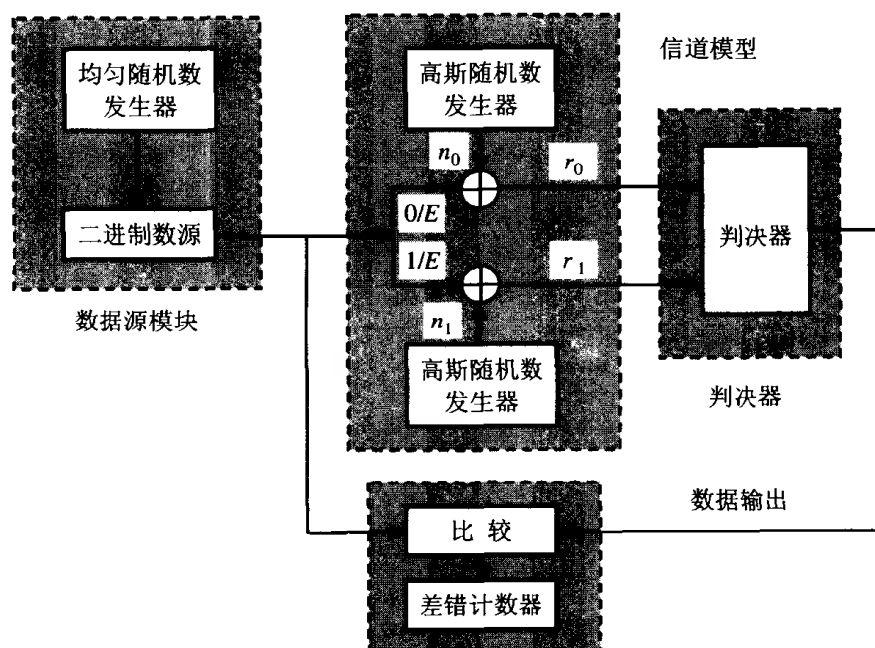


图 2-19 仿真系统模型

整个仿真的 MATLAB 源程序

```
smpe=zeros (1,13);
snr=0:1:12;
for n=1:13
    smpe (n) =pe_count (100000,n-1);
    % 计算不同信噪比下的误码率
end;
%theope=qfunct (sqrt (snr) );
plot (snr, smpe, '-*k', snr, theope, ':or');

function r=pe_count (toalbits,snr)
    % 计算在固定信噪比下的误码率
    inbits=rand (1,toalbits) >0.5;
    langada=sqrt (exp (-log (10) *snr/10) /2) ;
    s_snr=length (snr) ;
    n0=randn (1,toalbits) .*langada;
```

```

n1=randn(1,toalbits).*langada;
r0=1-inbits+n0;
r1=inbits+n1;
outbits=r0<=r1;
error=xor(inbits,outbits);
e_num=nnz(error);
r=e_num/toalbits;

```

图 2-20 是仿真的输出曲线，可以看出它和理论计算得出的结果是很接近的（有时间可以画出理论计算的曲线）。

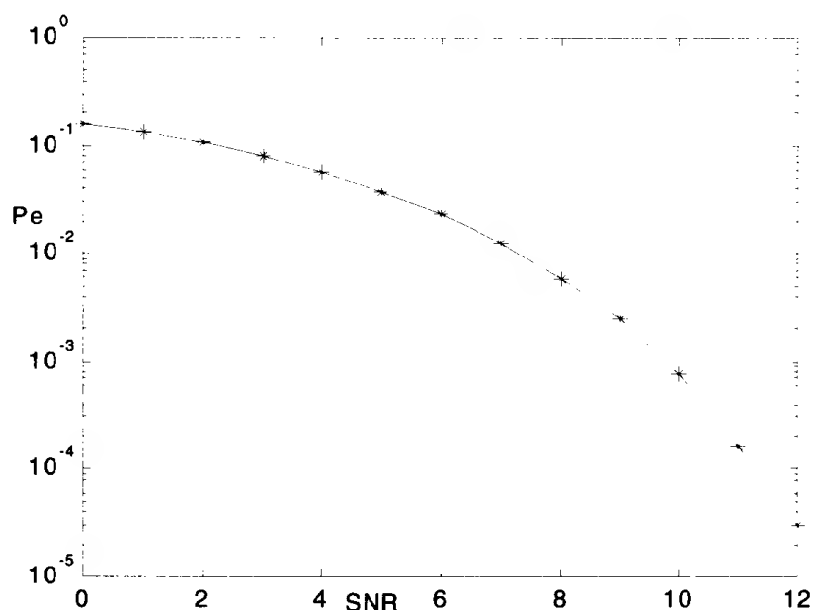


图 2-20 仿真的输出曲线

#### 4. 推广与改进

上面的仿真模型尽管是在二进制正交调制的前提下导出的，但其基本框架可以推广到更复杂的情况。例如，读者可以采取不同的信号波形，如双极性信号、单极性信号或者多电平信号。读者还可以对模型进行稍微修改，就可以将其应用于载波调制的数字通信仿真，甚至是现在最新的 CDMA 通信仿真，有兴趣的读者可以参考通信方面有关的书籍，来对这些系统进行仿真。

##### 2.4.2 排队系统仿真

上面的二进制通信仿真在本质上属于一种静态仿真，或者说是算法仿真。这类仿真在实现上相对较简单。动态仿真的程序设计则要复杂得多，它涉及到时间表示、时间的驱动等等繁琐的工作，从底层编起将是一个十分琐碎的工作。好在 MATLAB 为我们提供了一个动态系统建模和仿真的工具——Simulink，本书的后面章节将会具体介绍它的使用。有些简单动

态系统也可以用静态仿真的思想来实现，像排队系统就是一个例子，下面将讨论它的实现。

### 1. 排队系统简介

相较于前面的二进制通信，排队系统对读者而言应该要熟悉得多。排队现象在生活中无处不在，如顾客到在超市付款、病人在医院看病等等都要排队。此外，像计算机网络中数据的存贮转发、电话局的占线问题、交通枢纽的车船堵塞和疏导、水库的存储调节等等都是排队现象。可见排队系统是十分普遍的，而研究排队系统的意义重大。

数学上，研究排队系统的理论是排队论。对它进行抽象归纳，一个排队系统都有三个基本组成部分：

(1) 到达模式——指动态实体按怎样的规律到达，描写实体到达的统计特性。到达模式按顾客到来的方式可能是一个一个的，也可能是成批的；按相继到达的间隔时间可以是确定的，也可以是随机的；到达过程可以是平稳的也可以是非平稳。

(2) 服务机构——指同一时刻有多少服务设备可以接纳动态实体，它们的服务需要多长时间。它也具有一定的分布。服务机构除了指服务时间外，还应该考虑到不同的排列方式。

(3) 排队规则——指对下一个实体服务的选择原则。通用的排队规则包括先进现出（FIFO）、后进先出（LIFO）和随机服务等（SIRO）。

图 2-21 是排队系统的基本结构。



图 2-21 排队系统的基本结构

研究排队系统，人们关心的特性有：队列长度、顾客等待时间、服务机构空闲时间或服务律。

根据上面三个组成部分的不同可以对排队系统进行分类。用符号形式表示为

$$X/Y/Z$$

其中，X 代表到达间隔时间的分布；Y 代表表示服务时间的分布；Z 代表并列的服务设备的数目。表示相继到达时间间隔或服务时间的分布的符号是

- |               |                  |
|---------------|------------------|
| M——负指数分布；     | D——确定性；          |
| Ek——k 阶爱尔兰分布； | GI——相互独立的一般随机分布； |
| G——一般随机分布。    |                  |

例如，M/M/1 表示到达时间为负指数分布，服务时间为负指数分布，单服务设备的排队系统。

学过排队论的读者都可能会有这样的体会，排队系统的理论分析过程相当艰深、难懂。在简单情况下如 M/M/1，可以得到比较精确的结果，但很多情况只有近似结果。

计算机仿真是研究排队系统的另外一种方式。下面以 M/M/K 的仿真为例来介绍这种方法。

### 2. M/M/K 的计算机仿真

排队系统一种比较典型的离散动态系统，按照常规，它的仿真必然涉及到时间的表示，

事件驱动表示（到达时间和离开事件）。

这里，我们用一种静态仿真的思想来实现 M/M/1 的计算机仿真。

对排队系统中的每个动态实体的状态可以由三个量来反映：与前一个实体到达的时间间隔，在排到自己服务前的等待时间以及服务时间。其中服务时间和到达时间间隔服从指数分布，不受别的因素影响。开始服务前的等待时间则要排在前面的动态实体的状态影响。其更新算法可以表示

```
IF arr_interval (i) <= wait_time (i-1) +serve_time (i-1)
THEN
    wait_time (i) = wait_time (i-1) +serve_time (i-1) - arr_interval (i)
ELSE
    wait_time (i) = 0;
END;
```

其中，wait\_time (i) 表示实体 i 从到达起到开始接收服务的等待时间，serve\_time (i) 表示实体 i 接收服务的时间，arr\_interval (i) 表示 i 与前一个的到达间隔时间。算法表达了这样一个事实，如果某个实体在到达之后，发现处在它前面的动态实体已经结束服务，所以这个实体就不用等待，直接可以接收服务；反之，处在它前面的实体如果没有结束服务（包括没有开始），那这个实体等待的时间就是它前一实体结束服务的时刻减去它到达的时刻，也就是上面所描述的。

于是不难用 MATLAB 写出仿真程序

```
function [mwait_t,mstay_t,queue_l,use_rate]=smqueue (mean_arr,mean_lea,peo_num)
status = zeros (3,peo_num);
    % 用一个 3 行矩阵表示每个顾客的状态：
    % 到达时间间隔，等待时间，服务时间。
status (1,:) =exprnd (mean_arr,1,peo_num);
    % 随机生成各顾客的到达间隔
status (2,:) =exprnd (mean_lea,1,peo_num);
    % 随机生成各顾客的服务时间。

for i=2: peo_num
    if status (1,i) <= status (2,i-1) + status (3,i-1)
        status (3,i) = status (2,i-1) + status (3,i-1) -status (1,i);
    else
        status (3,i) =0;
    end;
    % 根据前面的法则对状态进行更新
end;
```

```

arr_time = cumsum (status (1,:) );
status (1,:) = arr_time ;
lea_time=sum (status) ;
stairs ([0 arr_time], 0: peo_num) ;
                                % 绘出各各顾客的到达时间图

hold on;
stairs ([0 lea_time],0: peo_num,'r') ;
                                % 绘出各各顾客的到达时间图

legend ('Arrive curve','leave curve',0)

hold off

figure;
plot (1:peo_num, status (3,:) , 'r', 1:peo_num, status (2,:) , 'k-') ;
                                % 绘出每个顾客的等待时间和停留时间

legend ('Stay Time','Wait Time',0) ;
n1=1;
n2=1;
mstay_t= (sum (status (2,:) ) +sum (status (3,:) ) ) / peo_num;
mwait_t= mean (status (3,:) ) ;
                                % 求平均停留时间和等待时间

queue_num= zeros (1,2*peo_num+1) ;
queue_time= zeros (1,2*peo_num+1) ;
n3=1;
                                % while 循环求每个时间队列的长度。

while n1<= peo_num
    n3=n3+1;
    if arr_time (n1) < lea_time (n2)
        queue_num (1,n3) = n1-n2+1;
        queue_time (1,n3) = arr_time (n1) ;
        n1=n1+1;
    else
        queue_num (1,n3) = n1-n2-1;
        queue_time (1,n3) = lea_time (n2) ;
        n2=n2+1;
    end;
end;
while n2<=peo_num
    n3=n3+1;

```

```

queue_num (1,n3) =peo_num-n2;
queue_time (1,n3) =lea_time (n2) ;
n2=n2+1;
end;
figure;
stairs (queue_time,queue_num,'k') ;
                                % 绘出队列长度的时间变化曲线
legend ('Queue Length Curve',0) ;
hold off;
temp=diff (queue_time) ;
overtime=max (queue_time) ;
queue_l=sum (temp.*queue_num (2: (2*peo_num+1) ) ) /overtime;
use_rate=sum (temp (find (queue_num) ) ) /overtime;
locy=max (queue_num) ;

```

这个程序有几个地方要注意:

(1) 首先因为  $M/M/1$  中, 各个实体的到达时间间隔和服务时间相互独立, 所以才能首先把每个实体的这两个时间, 根据所给参数产生模拟的随机数来。而在 MATLAB 里, 产生指数分布的随机数的参数是平均时间, 而非到达率, 这一点请读者注意。

(2) 确定排队系统状况的程序很简单, 只有极短的一部分, 程序的后面几块主要是计算该排队系统的队列长度随时间变化情况、平均队列长度和占用率。

(3) 计算某时刻队列长度的基本思想是用在此时刻前到达的总实体数, 减去在此时刻前结束服务的实体数。然后不难求出平均队列长度和占用率。关于具体编程请看注释。

### 3. 仿真结果

```
>>[w,s,l,r]=smqueue (0.5,0.4,200)
```

w =

1.3208

s =

1.7337

l =

3.6638

r =

0.8539

可以把计算机仿真的结果和理论推导的结果相比较, 会发现在实体数很大的情况下, 两者会很接近, 这也说明了理论推导结果的正确性。

图 2-22、图 2-23 和图 2-24 分别是仿真的结果。



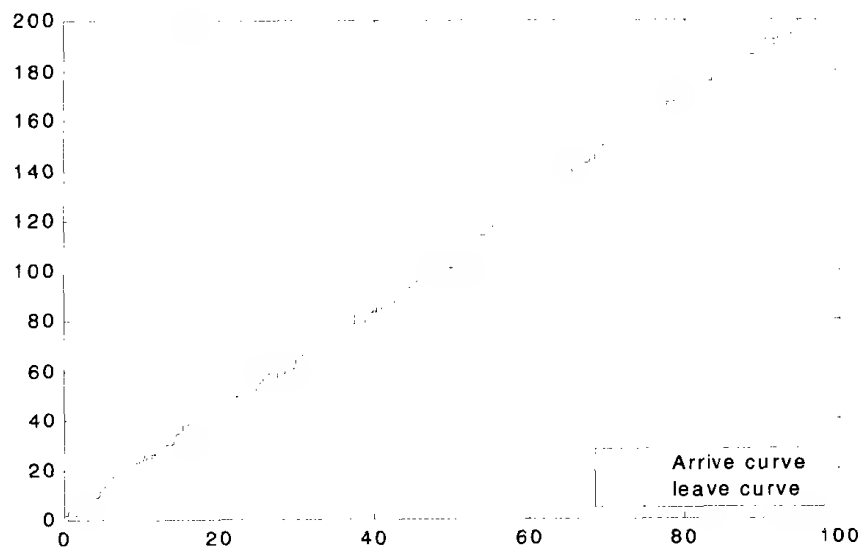


图 2-22 各顾客的到达时间和离开时间

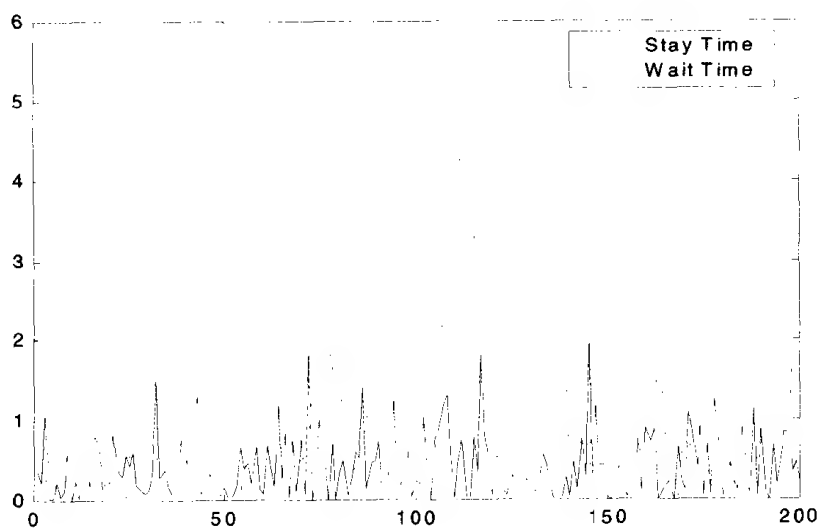


图 2-23 各顾客的停留时间和等待时间曲线

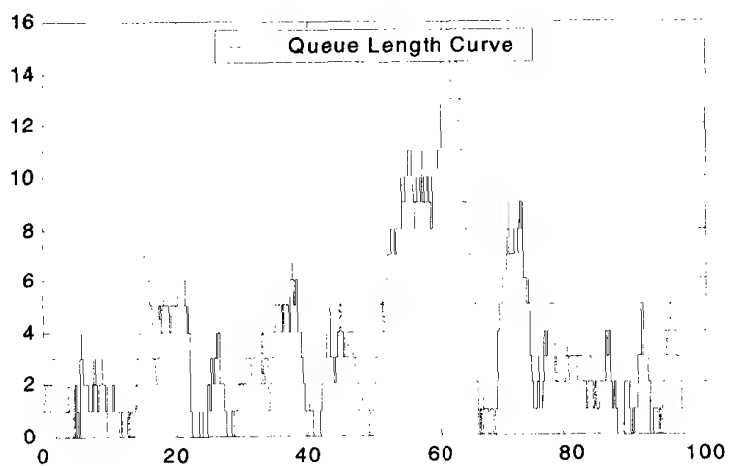


图 2-24 队列的长度变化曲线

#### 4. 仿真程序的改进和推广

尽管前面介绍的仿真程序是针对  $M/M/1$  的，但只需很小的改动就可以到其他情况。比如说，采用不同的随机数发生器，其他的部分保持不变就可以转换成  $G/M/1$ 、 $D/M/1$  或  $M/G/$  等等。

至于多服务台的情况，对更新算法则要做相应的改动。但思路基本上和单服务台的情况一样。

此外，前面的仿真程序是以实体的观察角度，程序的输入是实体的个数。可以把程序改成以时间为参数，这样就可以考察排队系统的稳态性能。

对于这些不同的情形，请读者参照排队论的书籍，仿照前面的例程编写仿真程序。



## 第三章 Simulink 入门

### 3.1 Simulink 简介

#### 3.1.1 什么是 Simulink

第二章涉及的仿真都是极其简单的静态仿真,为了处理更复杂的和时间有关的动态系统,读者就必须学习 Simulink 的使用。Simulink 是 MATLAB 提供的实现动态系统建模和仿真的一个软件包。它让用户把精力从编程转向模型的构造。随着学习的不断深入,读者会认识到 Simulink 一个很大的优点是为用户省去了许多重复的代码编写工作,用户就不必一步步地从最底层开始编起。

按照惯例,还是先给读者一点感性认识。Simulink 的最新版本是 Simulink4.0 (包含在 MATLAB6.0 里),启动 Simulink 的方法有很多种,按照 MATLAB 的传统方式,只要在 MATLAB 命令窗口输入

```
>> Simulink
```

这样,一个称为 Simulink Library Browser 的窗口就会出现在桌面上,它的样子如图 3-1 所示。

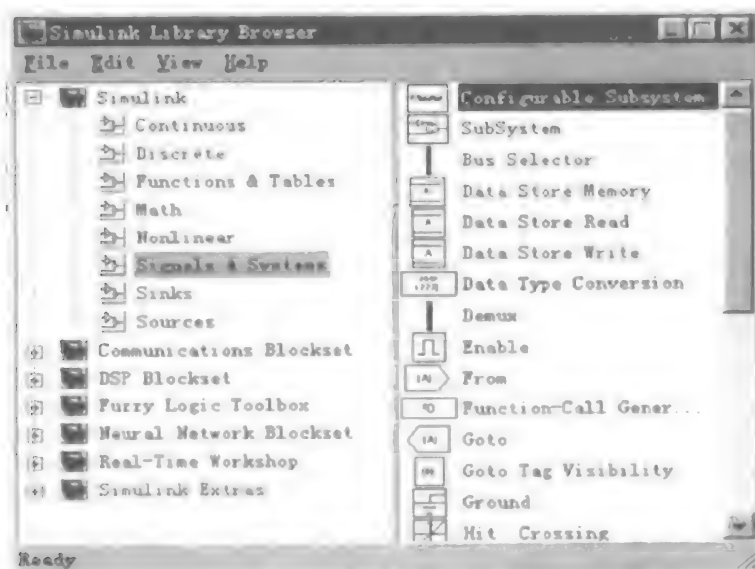


图 3-1 Simulink 库浏览器

读者也可以使用 MATLAB 主窗口的快捷按钮或者是 launch pad 子窗口里的命令来打开。如果读者使用的版本是 Simulink3.0 (包含在 MATLAB5.3 里),也不用担心,Simulink4.0 和 Simulink3.0 变化并不大。

Simulink 让人很振奋的一点就是支持图形用户界面，这比 MATLAB 传统的命令行方式要亲切得多。正是这样，读者就可以自己探索，不一定要按部就班地按本书的介绍来学习。当然先对 Simulink 工作原理有一个整体上的把握还是很有必要的。

前面讲过，Simulink 是用于动态系统建模和仿真的一个软件包，如果把这个过程比作建造房子，那么以前用高级语言或 MATLAB 语言编写仿真程序的方式就如同是从一堆沙开始造房子，这不但麻烦，而且有许多重复操作，建造者的精力会大量地浪费在一些相同的例如把沙变成砖块的事情上，以及如何把它们组在一起变成房子这些技术性的东西，而不能把更多的精力集中到房子的设计上。这体现在计算机仿真里，就等于是把精力过多的投入在某一个具体算法的设计上，而不是在模型设计构造本身。Simulink 的目的就是让用户能把更多的精力投入到模型设计本身。它首先提供了一些基本模块，这些模块就放在上面的库浏览器里，用户可以调用这些模块，而不必再从最基本的做起。Simulink 的每个模块对用户而言都是透明的，用户只需知道模块的输入输出以及模块功能，而不必管模块内部是怎么实现的。于是留给用户的事情就是如何连接这些模块来完成自己的仿真任务。连接的方式在 Simulink 里是很简单的，例如要连接两个模块，只需将一个模块的输入和另一个模块的输出用一根直线连起来就行了。模型构造好后，用户可以进行仿真，等待结果，或者改变参数，再运行。至于像各个模块在运行时如何执行，时间是如何采样（离散系统），事件是如何驱动等细节性问题，用户根本不需要操心，Simulink 都替你做好了。总之，Simulink 把那些最没有意思、最烦人的细节都屏蔽掉了，而留给用户一个友好的环境，让用户以最轻松、最有效的方式完成他们感兴趣的東西。

正如上面所说，库是存放 Simulink 模块的场所，图 3-1 所示的浏览器窗口就像是一个橱窗，用户可以在里面浏览并选择自己所需的模块，并可以调用（以后将会看到这只是一个拖动的操作）。下面对图 3-1 作一个简要的说明。

读者的窗口也许和图 3-1 显示的不太一样，这种现象很正常。Simulink 这一标题是必定会存在的，它是 Simulink 的基本模块库。而至于通信模块集(Communication blockset)，数字信号处理（DSP blockset）等等，视读者的安装而定了，你只有安装了这些工具箱，它们才会出现。但读者至少要安装实时工作间（real time workshop），因为这也是本书的主题之一。



图 3-2 Sinks 子库展开图

读者可以点击每个标题前的加号，看看库里面有些什么东西，就如在图 3-1 看到的一样，Simulink 下面分为：Continuous（连续）、Discrete（离散）、Functions&Tables（函数和平台）、Math（数学）、Nonlinear（非线性）、Signals&Systems（信号和系统）、Sinks（接收器）、Sources（源）等等子库。可见 Simulink 库是按功能分类的，用户在调用时会很容易就找到自己所需的模块。读者可以选择这些子库的一个，继续看看它里面的情况。图 3-2 显示的是 Sinks 子库里的模块。

从模块的名字可看出，Sinks 子库里的模块基本上是一些信号接收器，如 Scope（示波器）、Display（显示器）、XY Graph（XY 图形）等等。但如果你要满足你的好奇心，看看这些模块，双击这些模块不会起任何作用。

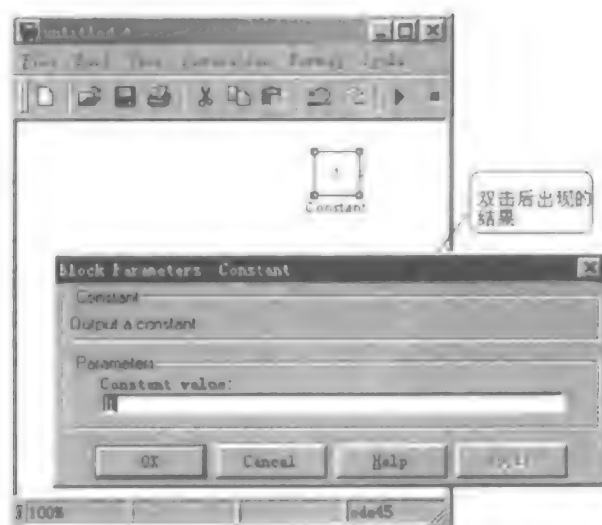


图 3-3 模型窗口及参数设置对话框

正确的操作是点击工具栏上最左边创建新模型的空白按钮，会出现一个如图 3-3 所示的窗口，不妨称为模型窗口，关于这个窗口下一节会详细说明。接下来的操作是回到浏览器窗口，在所感兴趣的模块上，比如 constant 模块，按下鼠标左键，然后拖动鼠标至新窗口才松开，这时新窗口会出现一个名为 constant 的方块。简而言之就是把模块复制到新窗口。图 3-3 显示了这时的情景，图中出现的对话框则是双击该模块后的结果。这个对话框的作用是为了让用户设置模块的参数，每一个 Simulink 模块都具有这种参数设置对话框。

创建模型的操作十分简单，读者完全可以独自摸索出来。要点是把你想要的模块拖到用户界面，然后按照你的模型，将它们的输入和输出端口用直线连起来。

### 3.1.2 Simulink 模型特点

下面，让读者体会一下用 Simulink 创建出的模型的样子，并分析它的一些特点。为此请在 MATLAB 的命令窗口，输入

```
>> thermo
```

将会出现图 3-4 所示的窗口，这个窗口和图 3-3 的区别在于前面的窗口没有包含任何模型，而这里包含了一个名为 thermo 的模型。这是 Simulink 的一个“房间热力学仿真演示”程序。

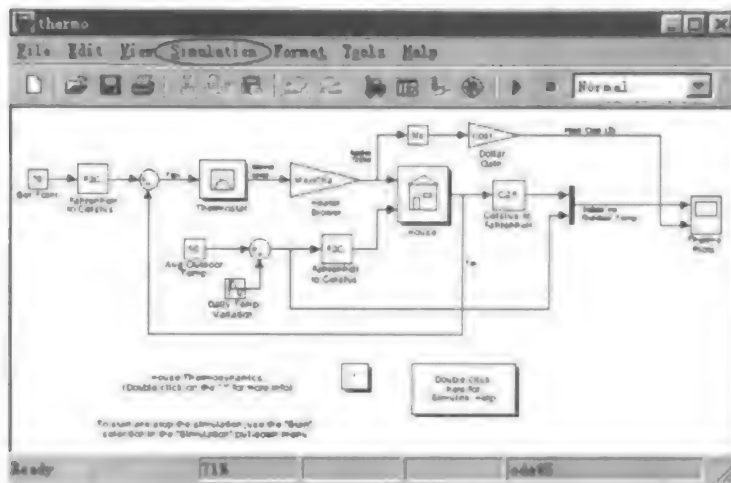


图 3-4 thermo 演示模型

读者可以点击执行按钮（见图 3-4），或者是 **Simulation** 菜单下的 **start** 命令来执行该演示程序。执行时，在窗口下面的状态条会显示出执行状况的进度。

然后，请双击 thermo 中名称为 thermo plots 的模块，这个模块其实就是 sink 子库里的 scope 模块。这样就可以看到仿真的结果——室内温度、室外温度和热量曲线。在 Simulink 里提供了许多类似于 scope 的可视化显示模块，而这种和实际示波器输出相似的图形化显示结果功能，正是 Simulink 的一个重要特性。

Simulink 所建模型的第二个特点是层次性。请读者随意双击用户界面上显示的模块。有些模块会显示出前面提过的模块参数设置对话框，然而有些模块则会出现图 3-5 所示的窗口，它就是双击名称为 house 的模块出现的情景。这个图表示 house 是由图右边所示的一些模块连接而成。像 house 这种由几个相互关联的模块组合而成的模块在 Simulink 里称为子系统，建立子系统的方法将在第四章介绍。而这种一个模块又由许多模块组成的特性，正是 Simulink 的层次性。为了和子系统相区别，这里把模型本身称为模型的顶层系统。层次性的好处是所建立的模型在结构上显得非常清晰整齐，让人一目了然。更主要的是，这个特性使得用户可以选择是采用从上至下建模，还是采用从下至上建模。因此，读者在建模时一定要层次，不要把所有的模块一股脑的摆在模型窗口的顶层，以致连线混乱，最后连自己都弄不清楚。

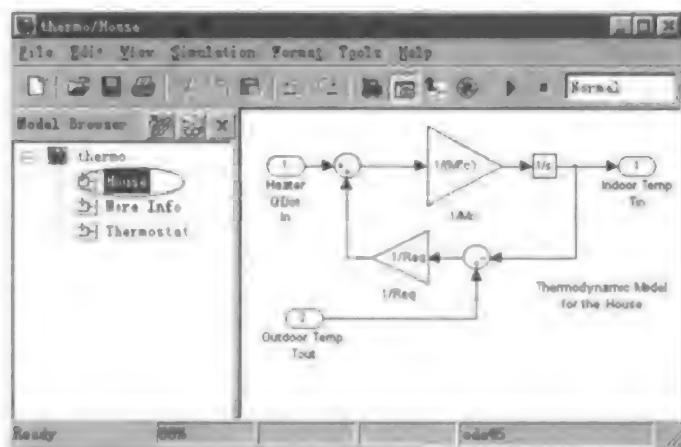


图 3-5 模型的层次性

在 Simulink 里，模型浏览器可以用来观看一个模型的层次结构。在图 3-5 的左侧，模型浏览器里的树形结构表示了 thermo 模型的层次结构。在最高层，即系统顶层（请用鼠标点击 thermo），这时右边的子窗口就显示系统层的连接情况。在 thermo 节点下的子节点，列出了系统层中所有的子系统，同样可以点击这些子节点观看它们的内部结构。thermo 是一个两层结构的模型，在更复杂的模型里，就可能会出现子系统内嵌子系统的情况。总之随层次的下降，模型的细节将会越来越具体。

第三个要提及的特点是，Simulink 为用户提供了一种封装子系统的功能，用户自定义该子系统的图标和设置参数对话框。还是以 house 子系统为例，它的房屋形状的图标就是封装后的结果，本书的第四章将会详细的介绍如何封装子系统以及封装后的好处。

除了上面提到的几个主要特点要仔细体会，读者还可以做很多其他的尝试，比如观察参数的改变对模型仿真结果的影响。

(1) Simulink 的 scope 模块可以提供一个或多个区域来显示信号的波形，而且用户可以改变每个信号的显示范围，放大或缩小信号的某个局部，以及设置 scope 的其他属性。读者可以按图 3-6 的提示，对 thermo plots 的坐标轴放大或缩小，或者是改变它的属性（见设置属性按钮），看看显示的图形会有什么变化。

(2) 改变标签为 set point 的常数模块的常数值参数，比如从 70 变到 80，看看仿真的结果有什么变化。这个模块的作用是设置预期的室内温度。读者也可以在仿真正在运行时，改变这个参数，看看仿真结果又有什么变化。

(3) 标签为 daily temperature variation 的正弦模块的作用是设置日温度变化，试改变它的幅度值参数，看看仿真结果的变化。



图 3-6 scope 示意图

除了 thermo 外，Simulink 还有许多其他的演示模型，读者可以直接在 MATLAB 命令窗口输入 demo，然后在列表框里找到 Simulink 选项获得。对于使用 MATLAB5.3 的读者，可以使用的另一种方式是在命令窗口输入

```
>> Simulink3
```



这时出现的 Simulink 模块库窗口是图标形式的，它和前面提到过的库浏览器的区别就像我的电脑和资源管理器的区别，为了和库浏览器相区分，我们把它称为模块库窗口，简称为模块库。至于选用哪一个界面，则要看个人喜好了，它们在功能上区别不大。图 3-7 是模块库的示意图。

本书倾向于使用模块库窗口，因为它里面的模块都用图标来表示，这样比较形象，而且诸如输入、输出端口的部分信息也能很直观的看出。

但在 MATLAB6.0，在命令窗口输入 Simulink4.0 则是一个错误的操作，因为 Simulink4.0 的库浏览器已经把这种图标形式的库窗口集成进来了。浏览器的左侧是浏览模块库层次的树形节点图，而当用户选中某个子库节点时，浏览器窗口的右侧就会显示该子库内包含的所有模块的图标。

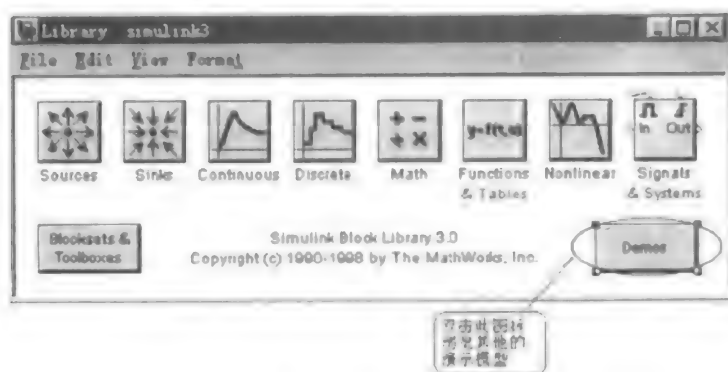


图 3-7 模块库窗口

## 3.2 创建一个简单的模型

这一节演示如何用 Simulink 创建一个简单的模型，其中的许多细节在前面一节已有提及，这里主要是把创建模型的整个流程梳理清晰。对那些通过自己的摸索，已经对这个过程已有了解的读者，建议最好还是粗略地浏览这一节，然后再学习后面的章节。

将要创建的示例模型所完成的功能是对一个正弦信号进行积分，并显示积分的结果。图 3-8 是最终建好的模型的样子。

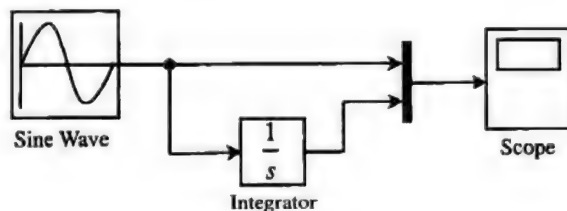


图 3-8 建好的模型

一般来讲，在模型结构都已经设计好的基础上，用 Simulink 建立模型的过程可以简单概括为：在 Simulink 的模块库中找到你所需的模块，并把它们拖曳到模型窗口中，将这些模块排列好，然后用直线把各个模块连接起来。具体的操作步骤为：

(1) 启动 Simulink 模块库浏览窗口（这当然是必须的）。

(2) 新建一个空白模型，为此请点击库浏览器工具栏上的空白按钮（见图 3-9）。在 Simulink 里，模型是保存在模型文件里的，新建一个空白模型，也就是新建了一个空白的模型文件，模型文件的后缀名为.mdl。也可以在模型窗口新建一个空白模型，其操作是使用 File 菜单下的 new a model 命令。

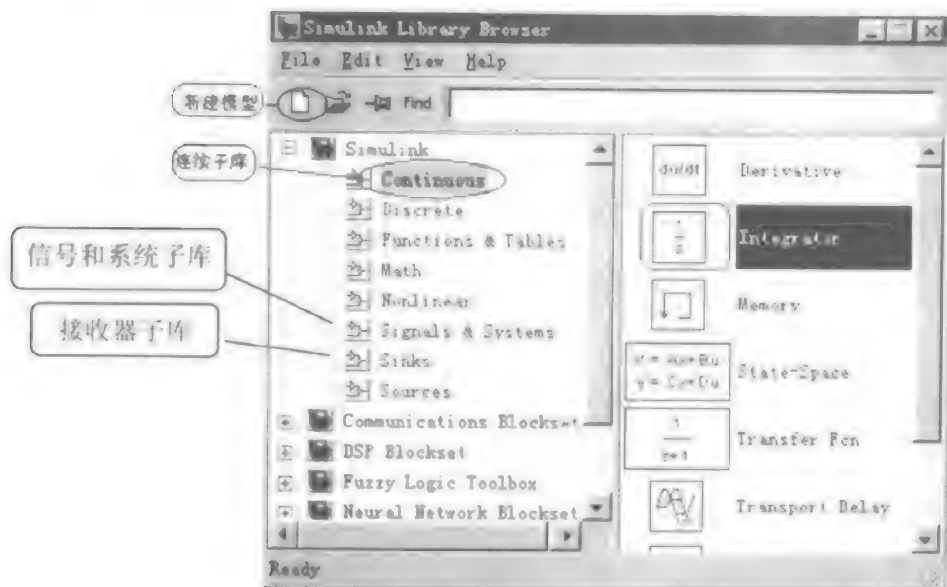


图 3-9 从模块库查找积分模块

(3) 在模块库浏览窗口中找到所需的模块。此模型包括的模块有：正弦波发生器、积分器、复用器和示波器。它们各自的位置分别是：正弦波发生器在 source 子库，积分器在连续系统子库（见图 3-9），复合器在信号和系统子库，示波器在接收器子库。

(4) 分别将所需的各个模块从库里拖曳到空白的模型窗口。这时，Simulink 会在模型窗口复制出这些模块。复制后的结果如图 3-10 所示。

(5) 将用户界面中的模块排列好，并按图 3-8 把它们用直线连接起来。注意模块的输入端只能和模块的输出端相连接，示意图见 3-10。

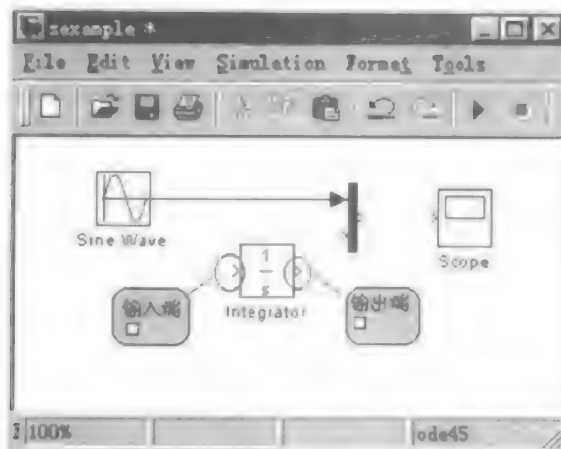


图 3-10 连线示意图

在图 3-8 所示的模型里，大部分的连线都是从输入端口到输出端口，只有一根直线从一条输出线连到 Mux 模块的输入端口，这种连线称为分支线。它表示一个模块的输出同时作为多个模块的输入。它的操作也很简单：

- 首先，把鼠标定位在 sine wave 和 mux 的连线上；
- 然后，按下 CTRL 键并保持，同时按下鼠标左键，并拖动鼠标至 integrator 的输入端（拖动时不要松开 CTRL 键），然后才松开左键和 CTRL 键。

图 3-11 描述了这个过程。

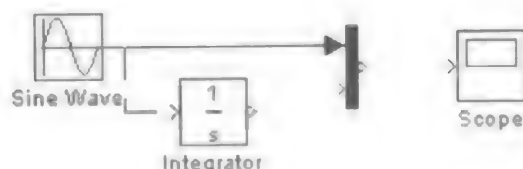


图 3-11 建立分枝连线的示意图

把所有的连线连好之后，这个仿真模型基本上就建好了。

- 注意：用户界面里不能有孤立模块存在，即不能有和别的模块没有任何连接的模块。模块的每个输入端，都要为它指定输入信号，即都要有连线。但输出端则可以空置。

现在，双击 scope 模块打开 scope 窗口，以观察仿真的输出波形。在开始仿真之前，先把仿真的时间设置成 10 秒。具体操作为，首先，从 simulation 菜单选择 parameter 项，打开设置仿真参数的对话框，见图 3-12。注意到 solver 页的 stop time 的值为 10.0，这个参数规定了仿真的结束时刻，它的缺省值就是 10.0。按 OK 按钮关闭对话框，所作的改动就生效了。和 Simulink3.0 的仿真参数对话框相比，Simulink4.0 增加了一个 advance 页。关于仿真参数对话框的详细说明，请见本书的第六章。

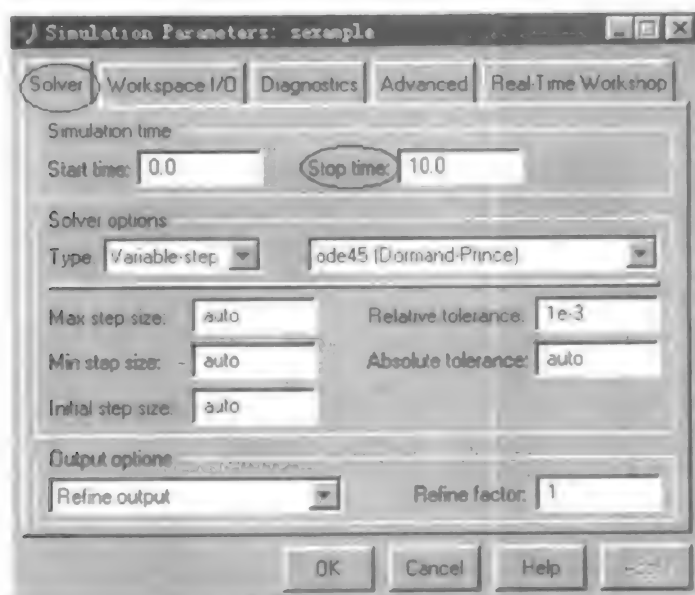


图 3-12 仿真参数设置对话框

运行仿真模型，就可以在 scope 里观察积分输出的波形，如图 3-13 所示。在 scope 的显示区域有两条曲线，一条是积分输出的曲线，另一条则是输入的正弦曲线。

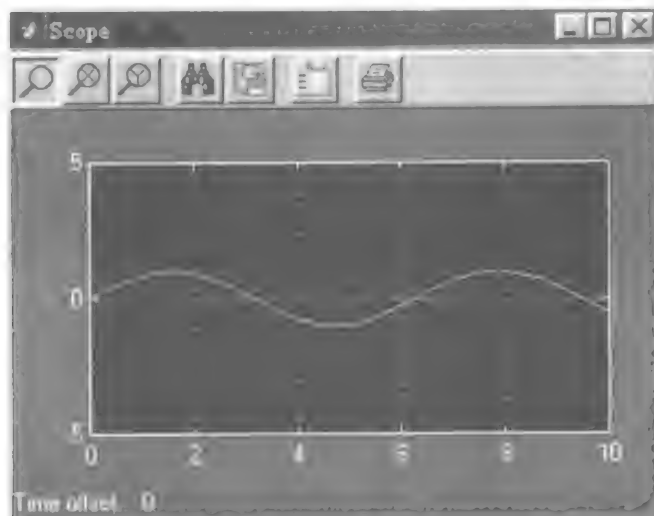


图 3-13 仿真结果的输出曲线

最后，选择 File 菜单下的 save 命令保存模型，模型文件的后缀名为.mdl。至此，用 Simulink 建立模型的基本操作就介绍完了。

### 3.3 熟悉 Simulink 模型窗口

用 Simulink 建立模型的操作除了将模块从库中复制到模型窗口之外，大多是在模型窗口完成的。读者要想熟练掌握这些操作，详细了解其中的各个菜单和按钮的功能是十分有必要的。

让我们从 File 菜单开始，模型窗口的 File 菜单和通常的 File 菜单没有什么大的区别，包括：new、open 和 save 等等常用的菜单项，分别用来新建一个模型，打开已有的模型以及保存当前模型等。需要注意的是 model properties 项，它的作用是设置模型的一些维护属性，比如模型的版本号、建立时间以及模型的描述等。另一个要注意的命令是 Source control 命令，这用来设置 Simulink 和源控制系统（SCS）的接口，所谓的源控制系统是管理文件的一种工具。而 Simulink 和 SCS 的接口，使得用户可以将模型文件从 Simulink 登记到 SCS 中，或者是从 SCS 中注销。

接下去的 Edit 菜单汇集了 Simulink 一些重要的操作。前面的一些项是基本的编辑命令：undo、copy、cut 和 paste，它们的作用和其他应用程序里的相同。在进行任何操作前，必须先选定操作的对象。在 Simulink 里，选定一个对象，只需用鼠标左键单击该对象即可。若要选择多个对象，可以使用两种不同的方法。

#### （1）逐个选择法。

选择时，按住 shift 键，然后再用鼠标左键逐个单击待选的对象即可，如果在一个已被选中的对象上单击，则表示放弃了这个对象。

## (2) 用方框选择多个对象。

这种方法比较简单，只要用一个方框把所有的待选对象包含起来即可。定义方框的操作很简单，首先，把鼠标指针移到一个空白处，然后按下鼠标左键，就定义了方框的起始角；然后拖动鼠标，把指针移到方框起始角的对角上。图 3-14 的左图表示了这个过程，而右图展示多个对象被选中后的状况。

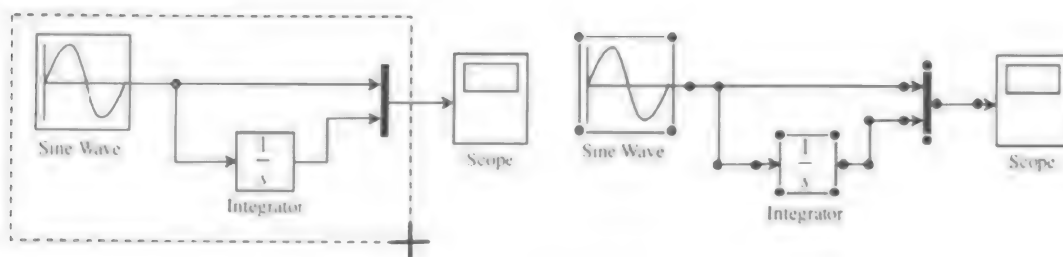


图 3-14 如何选中多个对象

读者可以自己尝试这些命令的使用方法，例如看看如何在模型内或模型之间拷贝模块，如何移动模块等。提醒一点，在进行这些操作之前最好先将模型另存一下。

Edit 菜单里的 **copy model** 命令的作用则是把当前的整个模型当成图片拷贝下来，作为其他的用途。比如，本书中关于模型的许多插图都是用这个命令得到的。

Edit 接下去的几个命令就比较重要了。在一般情况下，它们都处在被禁止的状态（为灰色）。这是因为不同的菜单命令有不同的适用对象。例如，**signal properties** 项是用来设置信号属性的，如果你当前选中的对象是一个模块，那这个菜单命令仍然处在不可用的状态。而它下面的 **model property** 却变成了可用状态。在 Simulink 里信号是和模块的端口以及与端口相连的直线相关的，因此只有选中了一条连线，**signal property** 才会变成可用。单击该命令出现的信号属性设置对话框如图 3-15 所示。

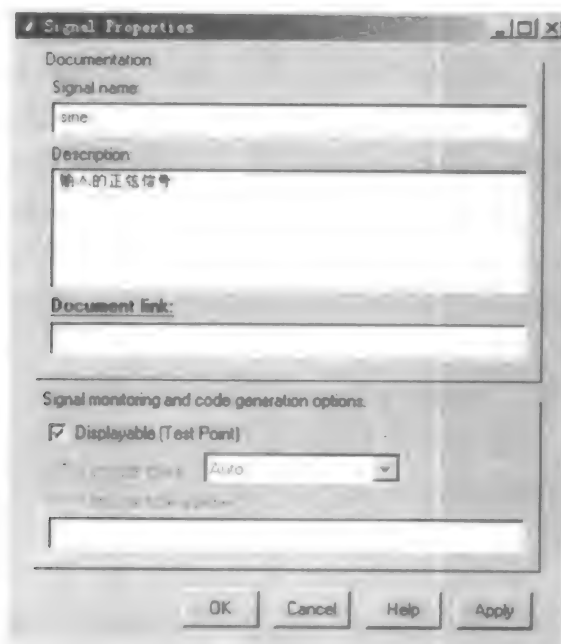


图 3-15 信号属性设置对话框

其中 **signal name** 属性的作用是标明信号的名称, 设置这个名称反映在模型上的直接效果就是与该信号有关的端口相连的所有直线附近都会出现写有信号名称的标签。图 3-16 反映了它的效果。

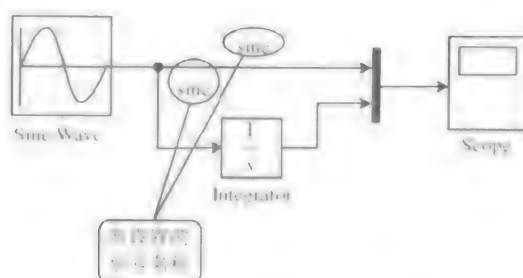


图 3-16 信号名称设置后的结果

这里, 设置给 **sine wave** 模块输出信号的名称为 **sine**。

**Block discription** 指该信号的描述信息, 比如功能、类型等, 建模者可以在上面写入有助于理解模型的任何说明信息。在建模时, 养成一个良好的注释习惯是十分必要的, 无论是对他人还是对建模者自己都是极有好处的。

**document link** 用户可以在里面输入一个指明与该信号有关的文档的 **MATLAB** 表达式, 例如你可以输入 `web(['file:/// which('foo_signal.html')])`。

信号属性对话框上的后面两个属性只有在使用 **RTW** 或生成 **C** 代码时才会用到, 本书的第 10 章将会讲到, 这里读者可以先把它忽略。

单击 **Edit** 菜单下的 **model properties** 命令可以设置模块的属性, 首先会打开一个模块属性设置对话框, 见图 3-17。

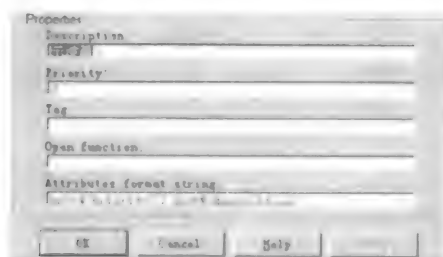


图 3-17 模型属性对话框

上图中各属性的作用在对话框上都有很详细的说明。这里只稍加说明一下。对于 **description** 属性, 给它设置一个值如图中的“示波器”, 要进行一些设置才能看出模型的变化。这里有两种方法, 一种是按上图所示, 在 **Attributes format string** 里输入 `pri=%<priority>;des=%<description>`, 按 **OK** 按钮之后, 模型图就呈现图 3-18 所示的样子。

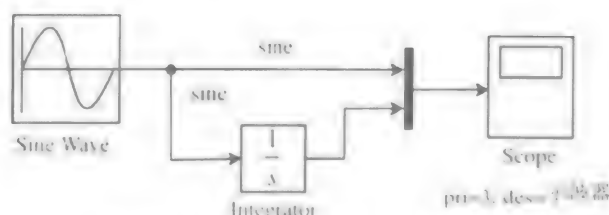


图 3-18 设置属性格式之后的模型图表

读者可以类似地把模块别的属性值标注到模块下面。

第二种方法是，在模块的数据提示里显示用户模块的描述字符串。所谓模块的数据提示，就是指当鼠标指针在模块上停留时，会出现的提示信息。需要做的设置是，把 View 菜单的 Block Data Tips 的 Show Data Tips 和下面的 User de 都置为被选状态。于是当鼠标指针在 scope 上停留时出现图 3-19 所示的提示信息。

priority 属性的作用是设置该模块在 Simulink 开始执行仿真模型时，相对于其他模块执行的优先级。

open function 属性是一个很有用的属性，它说明了在模块被双击后，Simulink 要调用的函数，这种函数在 MATLAB 里称为回调函数，关于它的概念将在后面的章节提及。

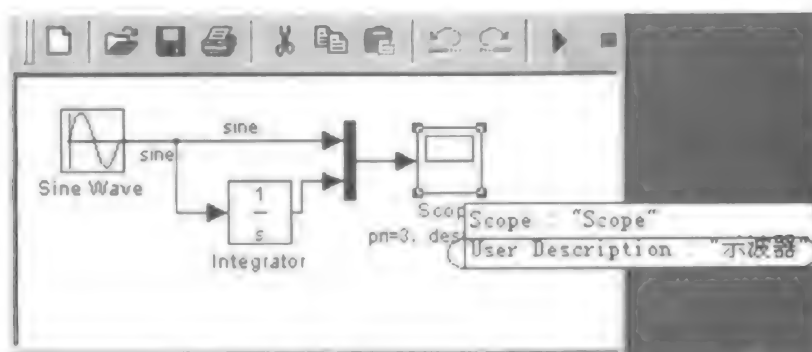


图 3-19 数据提示的示意图

接下去的 Create Subsystem 的作用是建立子系统。如何创建子系统以及如何封装子系统等内容将在下一章介绍。

Goto Library Link 一直到 unlock library 是关于库的一些操作，放在第四章介绍，这里就忽略。最后的 Update Datagram 作用是更新模型图表。它的作用是方便地把最近对库里的源模块（模型的模块都是库里的模块的一个拷贝）的任何改动，更新到现有的模型中，而不需要用户手动更新。关于它的详细说明也在第四章。

下面来看看 View 菜单。其中大多数的命令项读者可以自己试试它的作用。比较费解的一项是 Block Data Tips，其子菜单中的 Show Block Data Tips 一项选中和不选中的区别是当你把鼠标指针移到某一模块时有或没有模块的数据提示信息。而 Show Block Data Tips 下面的那些选项则说明提示信息里应该包含的信息。图 3-20 展示了 Show Block Data Tips 被选中后，含有模块的数据提示信息的样子。

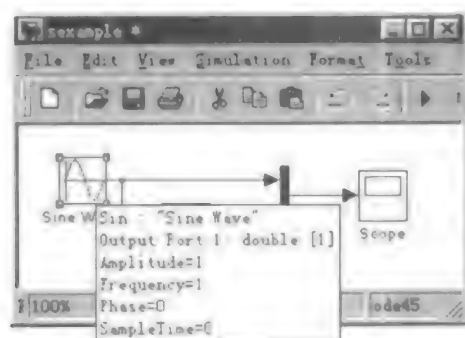


图 3-20 模块数据提示



Simulation 菜单下的 start 命令的作用是开始运行仿真。stop 则是终止仿真的运行，不管是否到达仿真参数里设置的终止时间。而 parameter 项，则是用来设置仿真参数的。最后面的两项，是相互排斥的选择项，它们表示 Simulink 的两种不同工作模式——正常模式和外部模式。请读者选择 normal 模式，至于外部模式将在第十章中提到。

Format 菜单下的命令的作用大多数都可以试出来，这里只强调最底下的几个选择项。图 3-21 就是所有的选择项都选中时，前面示例模型的样子。Simulink 里，几乎所有模块都支持标量或者向量的输入信号，也能产生标量或者向量的输出信号。例如图中的 mux 的输出信号就是一个两位宽的向量信号。要判定图表上哪条直线传递了向量信号，可以选中 Wide Vector Lines 选项，这时模型图表上所有传递向量信号的连线都会变宽，图 3-21 的 mux 的输出信号就是这种情况。而 Vector Line Widths 选项则在向量化的直线上标出向量的宽度，Port Data Types 选项的作用是标出模块中的端口的数据类型，如果是向量信号还指出它的宽度。

Sample Time Color 这一选项的作用是通过模块边框颜色来反映模块的采样时间快慢。由于在上例中，各个模块的采样时间都相同，所以选中这个选项，对图表外形的作用不明显。读者可以在 Simulink 里的 demos（演示模型）里选择一个模型来观看这个选项对图表外观的影响。关于采样时间的详细信息请看第五章。

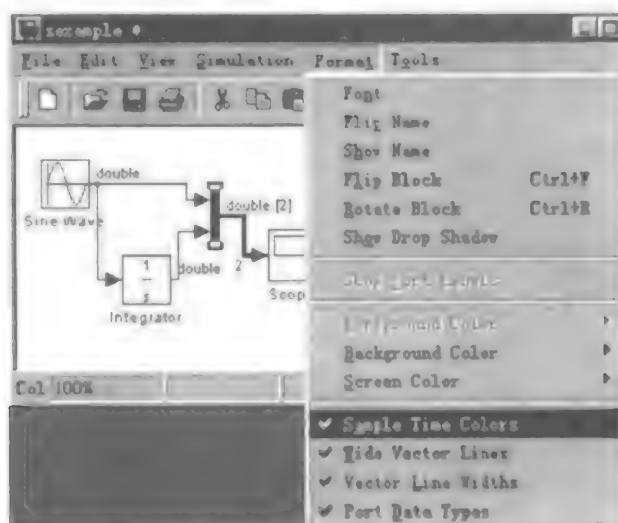


图 3-21 wide vector line 选项的效果示例

最后要介绍的是 Tools 菜单，这个菜单里的命令和 Simulink3.0 有比较大的变化，Simulink4.0 的许多新增功能都集中在这个菜单。其中，Simulink debugger 命令打开 Simulink4.0 新增的图形调试工具，而以往的 Simulink 调试功能都是基于命令行形式的（关于如何进行调试，请见第七章）。

而 Data Explorer 选项则是打开一个数据管理器，这个工具也是以前的 Simulink 版本所没有的。图 3-22 是在 Simulink 的演示模型 f14 下，它被打开后的样子（打开 f14 的方法是在命令窗口输入 f14）。数据管理器的作用是管理模型中定义和用到的一些数据变量，显示它们的数据类型，但无法添加变量或者修改变量的属性。

Tools 菜单的最后两项都是和 Real Time Workshop 有关的命令，本书将在第十章介绍它们。



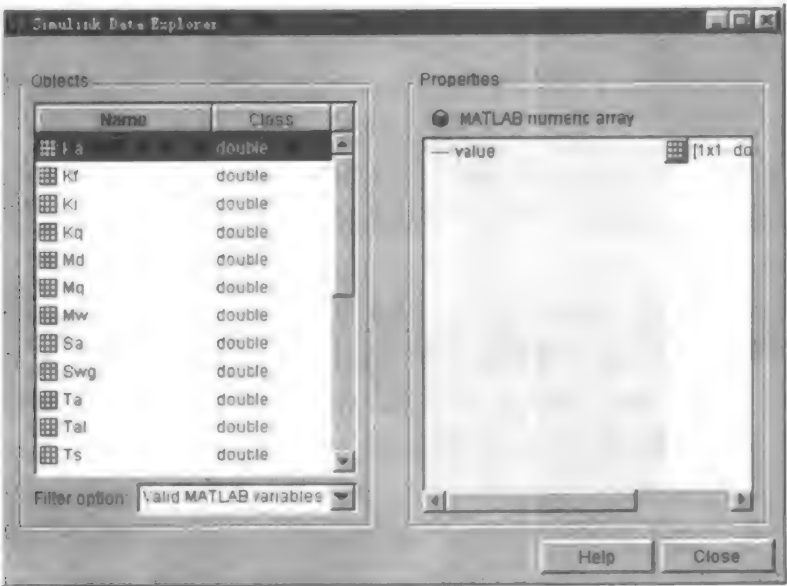


图 3-22 f14 下的数据管理器

### 3.4 键盘和鼠标操作总览

下面的这些表格汇总了 Simulink 里对模块、直线和信号标签进行各种操作的鼠标和键盘用法。

表 3-1 列出了一些关于模块的键盘和鼠标操作，这些操作仅在 Microsoft Windows 环境下适用，对于使用 UNIX 的用户，请看 Simulink 的用户向导。表 3-2 列出了关于直线的各种操作。

表 3-1 对模块进行操作	
任 务	Microsoft Windows 环境下的操作
选择一个模块	按鼠标左键
选择多个模块	shift+鼠标左键
从另一个窗口复制模块	拖动模块
移动模块	拖动模块
Duplicate	CTRL+鼠标左键，并且拖动鼠标，或者是鼠标右键和拖动
在模块间连线	鼠标左键
断开模块间的连接	shift+拖动

表 3-2 对直线进行操作	
任 务	Microsoft Windows 环境下的操作
选择一条直线	鼠标左键
选择多条直线	shift+鼠标左键

续表

任 务	Microsoft Windows 环境下的操作
画分枝直线	CTRL+拖动直线；鼠标右键并且拖动直线
用方框框主模块	按鼠标左键
移动直线段	拖动直线段
移动顶点	拖动顶点
建立直线段	shift+拖动直线

对信号进行标注以及在模型图表上建立描述模型功能的注释文字，是一个很好的建模习惯。读者请看下面的例子，来看看什么是信号标签和注释（图 3-23）。

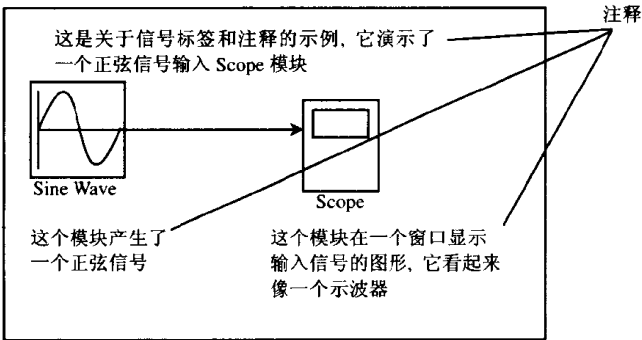


图 3-23 注释和信号标签示例

表 3-3 和表 3-4 分别列出了关于信号标签和注释的一些操作。给一个信号添上标签，只需用鼠标左键在直线上双击，然后输入文字就可。至于其他的操作，请读者看表 3-3。

表 3-3 对信号标签进行处理

任 务	Microsoft Windows 环境下的操作
建立信号标签	在直线上双击，然后输入
复制信号标签	CTRL+拖动标签
移动信号标签	拖动标签
编辑信号标签	在标签内单击，然后编辑
删除信号标签	在标签上 shift+单击，然后按删除键

建立模型注释的操作也很简单，只要在模型的空白处，用鼠标左键双击，然后输入注释文字即可。详细的命令说明请参阅表 3-4。

表 3-4 对注释进行处理

任 务	Microsoft Windows 环境下的操作
建立注释	在模型图表里双击，然后输入文字
复制注释	CTRL+拖动标签
移动注释	拖动标签

续表

任 务	Microsoft Windows 环境下的操作
编辑注释	单击文字，然后编辑
删除注释	按 shift 键 + 选中注释，然后按删除键

### 3.5 模块库简介

用 Simulink 建立仿真模型的过程，简单的就可以理解为将模块库里的模块拼搭在一起。因此读者必须了解其中的各个模块的作用，这一节就简单的介绍一下 Simulink 的库浏览器。

库浏览器的作用是让用户快速的对模块进行定位并且将它们复制到模型里去。用户可以像前面所讲述的浏览库树来定位模块，也可以通过查找功能来获得某个模块的位置。关于他们的操作读者可以查阅前面小节的内容，即使是自己摸索也是不难获得的。下面想重点说的是关于各个子库以及各个模块的功能。

库浏览器中模块的多少，或者根结点的多少，取决于用户的安装。但至少会包括 Simulink 节点和 Simulink extras 节点。读者点开（单击节点的加号，或者是双击名称）Simulink 节点，可以看到它包含下面这些子节点：

- （1）Sources 库，它包含了产生信号的模块，如 Sine wave（正弦波）和 Random Number（随机数发生器）等等。
- （2）Sinks 库，它包含的模块用于显示或者写模块的输出。如经常用到的 scope 模块，这个库里的模块还可以把输出写入文件中或者是保存变量到 MATLAB 工作空间。
- （3）Discrete 库，它包含了描述离散时间系统组件的模块。其中最典型的是 Discrete Transfer Fcn 模块（实现离散传递函数），Discrete Filter 模块（实现 IIR 和 FIR 滤波器）等等。
- （4）Continuous 库，包含了描述线性函数的模块。典型的代表有 Derivative 模块（输出输入信号的微分），Integrator 模块（对输入信号进行积分），State-Space 模块（实现线性状态空间系统）等等。
- （5）Nonlinear 库，包含的模块描述非线性函数。例如 Quantizer 模块（将一个信号按一定间隔进行离散化），Switch 模块（在两个信号间切换）。
- （6）Math 库，包含了描述一般数学函数的模块。其中的模块的功能就是将输入信号按照模块所描述的数学运算函数计算，并把计算结果作为输出信号输出。例如 Abs 模块（将输入信号取绝对值），Complex to Magnitude-Angle 模块（求一个复数信号的相位和模）等等。
- （7）Functions&Tables 库，包含了描述通用函数和查询表的模块。例如 S-Function 模块（把 S 函数文件和 Simulink 模型结合的模块，它的使用请看第九章），MATLAB Fcn 模块（将一个 MATLAB 函数或者是表达式作用于输入信号）。
- （8）Signal & Systems 库，它里面的模块允许复用信号（Mux）或者分离信号（Demux），实现外部的输入/输出（Inport 和 Outport），传递数据到模型中其他部分（From 等等），建立子系统（Subsystem，见第四章），以及其他的功能。

而 Simulink extras 库则包含了一些额外的模块, 用户自己建立的模块可以放在这个库。Simulink 还在 Blocksets and Toolboxes 库提供了一些专用的模块, 如用于通信的模块, 神经网络应用模块, 控制应用模块等等。读者可以通过在 MATLAB 命令窗口输入 Simulink3 命令打开模块库, 可以把上面的结构看得更清楚。在库浏览器里, Blocksets and Toolboxes 节点是不存在的, 而是把里面的各个子库直接作为结点。

下面就来详细介绍各个子库中模块的功能。

表 3-5 至 3-12 列出了各个子库中模块的简单说明。

**表 3-5 Sources 库**

模 块 名	目 的
band-Limited White Noise	把一个白噪声引入到连续系统中
Chirp Signal	产生频率增加的正弦信号
Clock	显示或者提供仿真时间
Constant	产生一个常数值
Digital Clock	按指定的间隔产生采样时间
Digital Pulse Generator	产生具有固定间隔的脉冲
From File	从一个文件读数据
From Work space	从在工作空间定义的矩阵读入数据
Pulse Generator	产生固定间隔的脉冲
Ramp	产生一个以常数斜率增加或者减小的信号
Random Number	产生正态分布的随机数
Repeating Sequence	产生一个可重复的任意信号
Signal Generator	产生多种多样的信号
Sine Wave	产生正弦波
Step	产生一个单步函数
Uniform Random Number	产生均匀分布的随机数

**表 3-6 Sinks 库**

模 块 名	目 的
Display	显示其输入信号的值
Scope	显示在仿真过程产生的信号的波形
Stop Simulation	当它的输入信号非零时, 就结束仿真
To File	写数据到文件
To Workspace	把数据写进工作空间里定义的矩阵变量
XY Graph	用一个 MTALAB 图形窗口来显示信号的 X-Y 坐标的图形

表 3-7

Discrete 库

模 块 名	目 的
Discrete Filter	实现 IIR 和 FIR 滤波器
Discrete State-Space	实现一个离散状态空间系统
Discrete-Time Integrator	离散时间积分器
Discrete Transfer Fcn	实现一个离散传递函数
Discrete Zero-Pol	实现一个用零极点来说明的离散传递函数
First-Order Hold	实现一个一阶保持采样-保持系统
Unit Delay	将信号延时一个单位采样时间
Zero-Order Hold	实现具有一个采样周期的零阶保持

表 3-8

Continuous 库

模 块 名	目 的
Derivative	输出输入信号的微分
Integrator	积分一个信号
Memory	输出来自前一个时间步的模块输入
State-Space	实现线性状态空间系统
Transfer Fcn	实现线性传递系统
Transport Delay	将输入延迟一给定的时间
Variable Transport Delay	将输入延迟一可变的时间
Zero-Pole	实现一个用零极点标明的传递函数

表 3-9

Nonlinear 库

模 块 名	目 的
Abs	输出输入信号的绝对值
Algebraic Constraint	将输入信号约束为零
Combinatorial Logic	实现一个真值表
Complex to Magnitude-Angle	输出一个复数输入信号的相角和模长
Complex to Real-Imag	输出一个复数输入信号的实部和虚部
Derivative	输出输入信号的时间微分
Dot Product	进行点积
Gain	将模块的输入信号成一个增益
Logical Operator	在输入信号实施一个逻辑操作
Magnitude-Angle to Complex	从模长和角度的输入输出一个复数信号
Math Function	实现一个数学函数
Matrix Gain	将输入乘上一个矩阵
MinMax	输出输入信号的最小和最大值

续表

模 块 名	目 的
Product	输出模块输入的乘积或者是商
Real-Imag to Complex	将输入信号作为是实部和虚部来成复数信号输出
Relational Operator	在输入上进行指定的关系运算
Rounding Function	实现一个舍入函数
Sign	显示输入信号的符号
Slider Gain	按一条斜线来改变标量增益
Sum	产生输入信号的和
Trigonometric Function	实现一个三角函数

表 3-10 Math 库

模 块 名	目 的
Fcn	将一个指定的表达式到输入信号
Look-Up Table	实现输入的线性峰值匹配
Look-Up Table (2-D)	实现两个信号的线性峰值匹配
MATLAB Fcn	应用一个 MATLAB 函数或表达式到输入
S-Function	访问 S 函数

表 3-11 Functions &amp; Tables 库

模 块 名	目 的
Backlash	对一个具有演示特性的系统进行建模
Coulomb & Viscous Friction	刻划在零点的不连续性
Dead Zone	提供一个零输出的区域
Manual Switch	在两个信号间切换
Quantizer	按指定的间隔离散化输入信号
Rate Limiter	限制信号的改变速率
Relay	在两个常数间切换输出
Saturation	限制信号的持续时间
Switch	在两个信号间切换

表 3-12 Signal &amp; Systems 库

模 块 名	目 的
Bus Selector	有选择的输出输入信号
Configurable Subsystem	代表任何一个从指定的库中选择的模块
Data Store Memory	定义一个共享的数据存储空间

续表

模 块 名	目 的
Data Store Read	从共享数据存储空间读数据
Data Store Write	写数据到共享数据存储空间
Data Type Conversion	将一个信号转换为另外一个数据类型
Demux	将一个向量信号分解输出
Enable	增加一个使能端到子系统中
From	从一个 Goto 模块接收输入信号
Goto	传递模块输入到 From 模块
Goto Tag Visibility	定义一个 Goto 模块标记的可视域
Ground	将一个未连接的输入端接地
Hit Crossing	检测过零点
IC	设置一个信号的初始值
Inport	为一个子系统建立一个输入端口或者建立一个外部输入端口
Merge	将几个输入线合并为一个标量线
Model Info	显示、修订控制模型信息
Mux	将几个输入信号联合为一个向量信号
Outport	为子系统建立一个输出端口，或者是建立一个外部输出端口
Probe	输出输入信号的宽度、采样时间并且/或者信号类型
Subsystem	表示在另一个系统之内的子系统
Terminator	结束一个未连接的输出端口
Trigger	增加一个触发端口到子系统
Width	输出输入向量的宽度

# 第四章 Simulink 详解

## 4.1 Simulink 的模块和模块库

### 4.1.1 Simulink 里的模块

模块是 Simulink 建模的基本元素，用 Simulink 建立仿真系统就是选用合适的模块，并用合适的方法连接模块。所以了解各个模块本身的作用，是熟练掌握 Simulink 的一个基础，读者可以根据第三章的提示自己进行摸索，也可以在第八章找到部分模块的参考说明。在 Simulink 里，模块的基本信息可以通过模块数据提示来显示，Simulink 还允许读者自定义数据提示信息的显示内容，对应的方法在上一节里已经有了介绍。这里将涉及的是一些基本概念，这些概念有助于加深读者对模块的理解。

#### 1. 虚模块和非虚拟模块

Simulink 里的模块可以分成最基本的两类：非虚拟模块和虚模块。非虚拟模块在一个系统的仿真中扮演一个重要的角色，向模型里增加或删除一个非虚拟模块，就会改变仿真的行为。而虚模块在系统里则扮演一个次要的角色，虚模块的作用仅仅是有助于用户图形化地组织好模型，而对模型的行为没有任何贡献。但是这种划分也不是绝对的。有些模块在某些场合是虚模块，在另外一些场合却是非虚拟模块，这种模块在 Simulink 里被称为条件虚模块。

表 4-1 列出 Simulink 里常见的虚模块以及条件虚模块。

还是以第三章建立的简单模型作为示例来说明两者的区别。它的模型图表如图 4-1 所示。

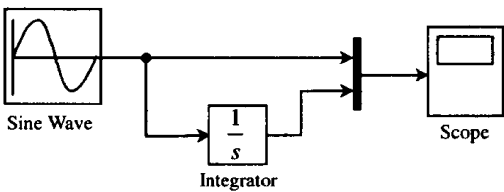


图 4-1 模型图表

表 4-1 常见的虚模块以及条件虚模块

模块名称	成为虚模块的条件
Bus selector	总是虚模块
DataStoreMemory	总是虚模块
Demux	总是虚模块
EnablePort	总是虚模块
From	总是虚模块



续表	
模块名称	成为虚模块的条件
Goto	总是虚模块
GotoTagVisibility	总是虚模块
Ground	总是虚模块
Inport	总是虚模块，除非它存在于条件子系统内，并且和一个 outport 模块直接相连
Mux	总是虚模块
Output	当它存在于任何子系统内（条件或者非条件）它都是虚模块，但处在模型的最顶层时（也即模型系统里），就不是虚模块了
Selector	总是虚模块
Subsystem	如果该模块不是条件执行的，它就是虚模块
Terminator	总是虚模块
TestPoint	总是虚模块
TriggerPort	当输出端口不存在时就是一个虚模块

在图中黑框代表的模块就是 mux 模块，它就是一个虚模块。mux 模块的作用就是将输入的多条信号线合为一个向量信号（向量信号的概念见后），这在实际的物理模型里，就像是把几根独立的信号线用绳子扎起来，但并不会改变信号线所传递的信号。这个模型，其实和下面的模型（图 4-2）等价。

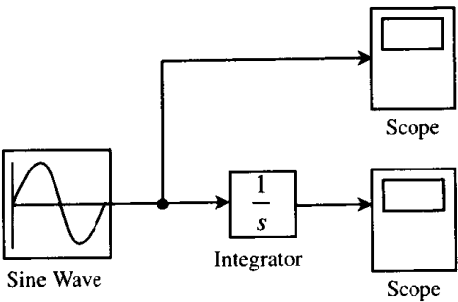


图 4-2 等价图

mux 模块在图形组织上的作用就是减少了连线的数目和 scope 模块的数目，只用一个 scope 模块在一个窗口对照地显示两个波形，这样使模型显得很有条理。

2. 标量信号和向量信号

向量信号和标量信号的概念在上一章曾粗略地提及。从字面上不难理解，向量信号是多个信号的集合，它对应着实际系统中几条并行连线的合成。在 Simulink 里，充当信号传递的直线只是一个图形化的虚拟形式，实际上信号的传递是依赖变量的，那么向量信号实际上就是指它的输出（输入）变量是一个向量，同理也就不难理解标量信号了。

缺省情况下，大多数模块输出的都是标量信号。对于输入信号，模块都具有一种“智能”的识别功能，能自动进行匹配，这是 MATLAB 语言的一贯风格，实现起来十分简单。读者在学习完本书第九章关于 S 函数编写那一章后，就能切身的体会到这一点。

那如何让 Simulink 的模块输出向量信号呢？很简单，只需要设置一下模块参数。还是以

前一章建立的那个模型为例，下面的操作将会让 Sine Wave 产生一个向量信号。

请双击 Simulink 用户界面里的 Sine Wave 图标，打开它的参数设置对话框。按图 4-3 所示设置参数。

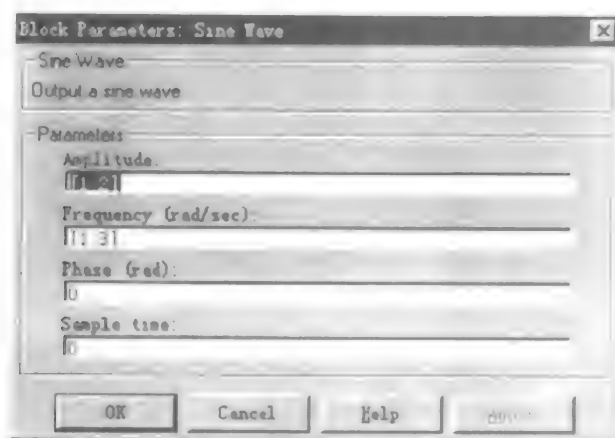


图 4-3 设置 sine wave 的参数

点 OK 按钮关闭参数设置对话框后，让模型图表显示向量信号的宽度，就可以看出 Sine Wave 的输出端是一个两位的向量信号。图 4-4 描绘了它的样子。

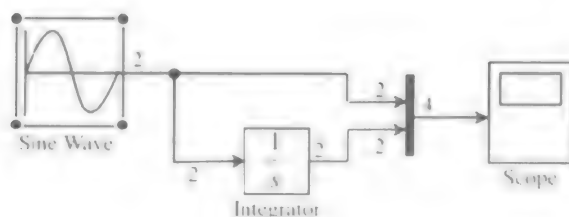


图 4-4 Sine Wave 输出了两位的向量信号

图 4-5 是参数设置改变后获得的仿真结果。从图中可以看出，Sine Wave 输出了一个两位的向量信号，即包含了两个幅度和频率均不同的正弦信号。这个向量信号和它经过积分后获得的向量信号被虚模块 mux 合并为一个 4 位的向量信号，这个向量信号在 scope 里的显示波形就是图 4-5 所示的。

这个例子说明了，Sine Wave 模块输出的信号宽度受参数的控制。当参数设置为一个向量时，例如上面的[1 2]，它表明该模块输出的第一个正弦信号幅度为 1，第 2 个信号幅度为 2，也就是输出了一个两位的向量信号。但是要注意，模块中所有的参数都是按向量元素的位置来和信号对应，不存在自由组合的问题。例如上例中的幅度和频率分别设置成[1 2]和[1 3]，这样不是说产生四个信号，而是两个信号。它们的参数如果用[幅度 频率]表示，则分别是[1 1]和[2 3]。此外，在设置向量参数时，一定要注意不同参数之间向量大小的匹配问题。Simulink 有时并不要求所有的参数都设置成同样大小的向量，例如，幅度[1 2]，频率[1]是可以的。但[1 2 1]和[1 2]的搭配却会出现错误提示，读者可以自己体会。

以上说的只对像 Sine Wave 那样的数据源模块才适用，它们的输出完全视设置的参数而定。但 Simulink 里的更多模块是既有输入端又有输出端的，一般它们的输出信号受输入信号和参数共同控制，例如 demux 虚模块。



图 4-5 参数改变后的仿真输出图形

Simulink 有很多模块具有多个输入端，如果使用时，读者把标量信号和向量信号混合输入，例如 sum 模块的两个输入端，一个是标量信号，另一个是向量信号。这时 Simulink 会怎样处理呢？下面建立一个简单的模型做一下实验。

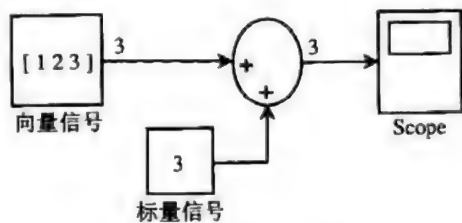


图 4-6 混合输入试验模型

图 4-6 是该模型建好后的图表。表 4-2 列出了模型所需的模块以及各自的参数设置。

表 4-2 所需模块及参数设置

图标标签	模块类型	所在子库	参数设置
向量信号	constant	Source	Constant value [1 2 3]
标量信号	constant	Source	Constant value 3
	sum	Math	缺省设置
scope	scope	Sink	缺省设置

运行仿真，会发现 scope 的显示区域出现了三条直线，其幅度分别为 4、5、6，也就是说 sum 的输出是一个向量信号。

这个现象在 Simulink 里称为输入信号的标量扩展，即把输入标量扩展成和向量信号同样大小的向量，新扩展的向量信号的元素就是原来的标量信号。注意，这种处理不能推广到两个不同长度的向量的混合，因为这时候 Simulink 无法决定扩展的方法。例如一个是两位的向量信号，一个是三位的向量信号，Simulink 对两位的向量信号扩展时无法决定是用第 1 位还是用第 2 位。这个道理也可以解释设置模块参数为向量时遇到的匹配问题，其实，也可以把设置参数为向量理解成一种扩展。

在 Simulink 里，还有一种称为参数的标量扩展。请看图 4-7 所示的例子。

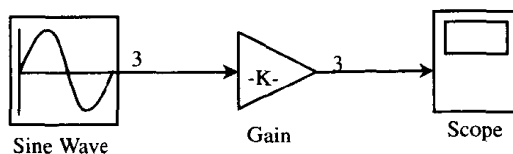


图 4-7 参数的标量扩展

图 4-7 中，Sine Wave 的幅度属性设成[1 2 3]，而 gain 模块的 gain 属性也设置为[1 2 3]，它的意思是输入向量信号的三个信号按顺序分别放大 1、2 和 3 倍，你运行模型可以看到这个结果。当 gain 属性设置成一个标量，例如 1 时，这时 Simulink 自然会将这个参数扩展成和输入同长度的向量，这里是[1 1 1]。这就是所谓的参数标量扩展，它其实是很简单的概念。

并不是所有的模块都会完成这种标量值向向量值的转换，关于这方面的信息请看本书附录的模块参考。

#### 4.1.2 Simulink 的模块库

库技术使得用户能从外部库里拷贝到自己需要的模型，并且在库里的模块变动时能够自动地更新用户先前已拷贝在模型里的模块，而不用用户手动更改。为了更好地理解 Simulink 库的这种能力，我们先介绍几个术语。

库——模块的集合；

库模块——库里的模块；

引用模块——库模块的拷贝；

链接——引用模块和库模块之间的连接，它使得 Simulink 在库模块发生变化时，能够更新引用模块；

拷贝——从库模块或者别的引用模块产生一个引用模块的操作。

图 4-8 反映了这些术语间的关系。当用户从库里用拖曳操作将库模块拷贝到用户的模型里时，Simulink 会在模型里建立一个参考模块。用户可以设置参考模块的参数，但不能封装（关于封装，读者请看 4.5 节）参考模块，对于封装好的模块，用户也不能修改封装。注意，Simulink 不是简单地复制出一个参考模块，它同时还建立了一个从参考模块到它对应的源库模块的链接。这个链接的作用是在参考模块里保存了它的源模块的信息，以便在用户更新图表时，Simulink 能按照模型中引用模块的保存的信息，搜寻它对应的库模块，并根据它重新产生一个参考模块来替代原模块。

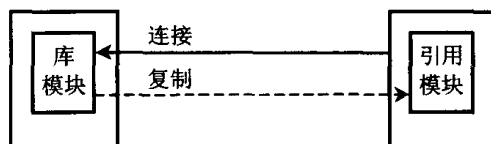


图 4-8 库模块和引用模块间的关系

确切地说，参考模块和库模块是靠名称链接起来的，也就是参考模块所链接的特定模块在进行复制时的名称才有效，这个意思是很明白的。

当 Simulink 试图更新参考模块时，如果它在 MATLAB 路径上找不到库模块和源库，那

么这个链接将变成无法确定的。Simulink 产生一个错误信息，并将这些参考模块用红色的虚线表示。图 4-9 显示了链接无法确定的情形。

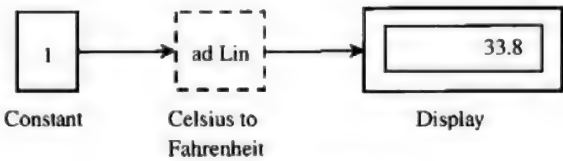


图 4-9 链接不确定的情形

读者可以按照下面的步骤来获得这个效果。首先，新建一个模型，分别从 source、Simulink extras 和 sink 库里取出相应的模块，按图 4-9 把线连好，最后别忘了保存模型；然后按照图 4-10 所示进行操作。

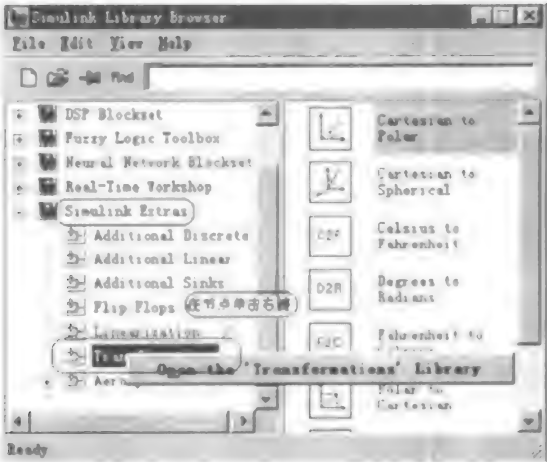


图 4-10 打开 transformations 库

在 Transformations 节点按右键弹出菜单 “Open...”，用左键单击此命令，打开 Transformations 库，如图 4-11 所示。

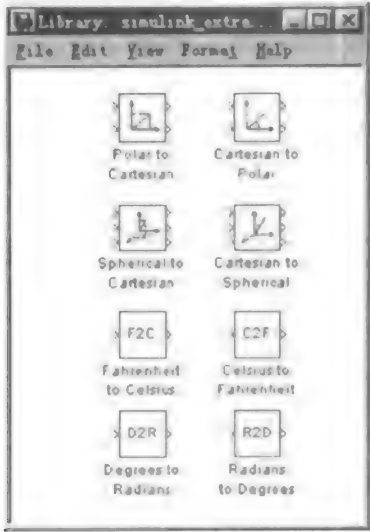


图 4-11 transformations 子库

读者请先将库另存为其他名称的库作为备份，这个操作将另存整个 Simulink\_extras 库，而不仅仅是 Transformations 子库。读者在另存时，最好是不要改变目录，这样万一出现错误时，就可以把库复原。复原的方法很简单，用备份库覆盖已经损坏的 Simulink\_extras 库，并把它文件名改为 Simulink\_extras；然后用 Edit 菜单下的 Unlock Library 命令，使库成为可编辑状态，接着把 Celsius to Fahrenheit 模块删除，再把库保存。完成这些操作后，再回到原来的模型窗口，单击 Edit 菜单下的 updat datagram 项，就会出现图 4-9 所示的内容。

可以按下面的几种方法来修复无法确定的链接：

(1) 删除未链接的参考模块，将库模块重新复制到模型里。

(2) 把包含所要求模块的目录加到 MATLAB 的搜索路径里，然后再从模型窗口选择 Update Datagram 命令更新图表。

(3) 双击参考模块，在弹出的设置对话框里，改正路径，按 Apply 按钮关闭对话框。

对于前面所举的例子，由于库模块已经被删除，要修复无法确定的链接只能重新建立一个库模块，然后将它重新和参考模块对应起来。但在这里，只需要用备份库复原原来的库即可。

在 Simulink 里，用户可以通过获得模型的库信息来查询模块的链接状况。库信息包括下面几个部分：

- 模块——模块路径；
- 库——库的名称；
- 引用模块——所引用的模块路径；
- 链接状态——表示链接的状态，确定或无法确定。

要获得该信息，可以用下面的 MATLAB 命令。

```
>> libdata=libinfo(' thermo')
```

```
libdata =
```

```
3x1 struct array with fields:
```

```
Block
```

```
Library
```

```
ReferenceBlock
```

```
LinkStatus
```

Libdata 是一个结构。可以用这样的格式访问结构的字段：结构名. 字段名。例如要查看 Block 字段值可以输入

```
>> libdata.Block
```

```
ans =
```

```
thermo/Celsius to
```

```
Fahrenheit
```

```
ans =
thermo/Fahrenheit
to Celsius
```

```
ans =
thermo/Fahrenheit
to Celsius
```

这个结果说明，在 **thermo**（前面提到的演示模型）里有三个模块是参考模块，**Block** 字段显示了参考模块的名称。

注意，**MATLAB** 里的命令和变量名都是大小写敏感的，所以在输入上面的变量时一定要注意大小写。比较好的办法是直接拷贝过来。

下面的注意力将集中到链接状态上。

链接状态可以取下面几个值：

- 'none'，表示该模块不是一个参考模块；
- 'resolved'，表示该模块是参考模块并且链接是确定的；
- 'unresolved'，表示该模块是一个参考模块，并且链接是无法确定的。

可以通过 **Edit** 菜单下的 **Break Library Link** 来去掉参考模块的链接，这样库模块的任何变动将不会影响用户模型中的模块（此时，已经不是参考模块了）。

或者用下面的 **MATLAB** 命令实现，

```
>>set_param('linkexample/')
```

这个命令的意思是把链接状态设置成 'none'，也就是使模块不再是参考模块。注意这个过程是不可逆的，你不能把一个不是参考模块的模块变为参考模块，因为模块原来保存的信息消失，**Simulink** 不知道链接到哪个模块。

同 **MATLAB** 语言一样，**Simulink** 也提供了扩展系统的能力。它允许用户生成自己的库，用户可以把一些常用的模块放在一个库里，免得寻找时麻烦。在后面的章节里将会教会读者创建自己的模块以及库。

选择 **Simulink** 用户界面 **File** 菜单下的 **New Library** 命令，弹出一个空白的库窗口。然后把读者常用的模块复制到该窗口。因为你复制到新建库里的模块也仅仅是参考模块，为此选中所有的模块 (**Ctrl+A**)，用 **Edit** 菜单下的 **Break Library Link** 去掉所有参考模块的链接，这样它们就可以作为库模块使用了，最后保存所建的库。图 4-12 是建好的库的示意图。

但是这个库有一个缺点：它不能用库浏览器浏览，在下一节将会介绍如何自定义能用浏览器浏览的库的方法。

以上讲述的命令操作是基于 **Simulink3.0** 的，在 **Simulink4.0** 里，与参考模块和库模块间的链接相关的操作有一些细微的变化。具体体现在 **Edit** 菜单，用 **Link Options** 子菜单替代了 **Break Library Link** 命令。而 **Break Library Link** 的功能用 **Link Options** 子菜单里的 **Disable Link** 命令来完成，**Link Options** 里还提供一个 **Restore Link** 命令用于恢复被禁止的链接，这是

Simulink4.0 新增的功能。



图 4-12 库浏览器

Simulink4.0 新增的另外一个功能是传递链接修改。前面说过, 具有激活的链接的模块不能具有结构性的变化。当用户试图恢复一个具有结构性变化的链接时, Simulink 会提示用户是传递还是放弃这个变化。如果用户选择传递, Simulink 会根据修改后的参考模块来更新库模块, 也就是提供了一个参考模块的反作用。如果用户选择放弃变化, Simulink 会用库模块替换已被修改的参考模块。当用户要恢复一个具有非结构性变化的链接, Simulink 不用提示是传递或放弃变化, 而直接将链接激活。

用户也可以通过 Link Options 子菜单里的 Propagate/Discard Changes 命令来手动地确定对一个参考模块所作结构性变化是传递还是放弃。使用这个命令, 当然要先选择一个参考模块。要想看参考模块和库模块之间的非结构性参数变化, 可以选择 Link Options 子菜单里的 View 命令。

## 4.2 模拟方程

如何模拟方程是 Simulink 初学者最困惑的问题之一, 在这一节将给出一些这方面的例子, 希望能加深读者对它的理解。

### 1. 将摄氏温度转换为华氏温度

第一个例子是模拟把摄氏温度转换为华氏温度的方程:

$$T_F = (9/5) T_C + 32$$

首先, 来考虑一下所需要的模块:

- (1) Ramp 模块, 用来产生温度信号, 所处库为 Source 库;
- (2) Constant 模块, 用来产生一个常数 32, 同样来自 Source 库;
- (3) Gain 模块, 将输入信号乘上 9/5, 来自 Math 库;
- (4) Sum 模块, 把两个量加起来, 也来自 Math 库;
- (5) Scope 模块, 显示输出的结果, 来自 Sink 库。

然后, 把所有的模块复制到模型窗口中。图 4-13 显示了这个情形。



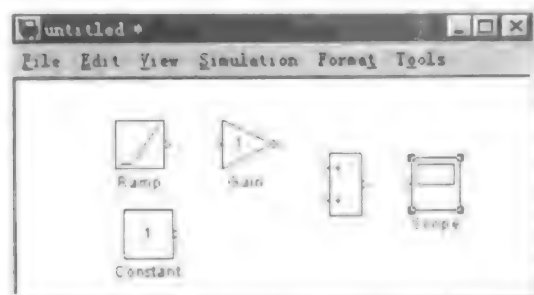


图 4-13 将所需模块复制到模型窗口

给 Gain 模块和 Constant 模块设置合适的参数值 (Gain 为 9/5, Constant 为 32), 就是双击模块方框, 打开参数设置对话框输入相应的数值。下面, 就可以连接各模块了, 请读者按照图 4-14 进行。

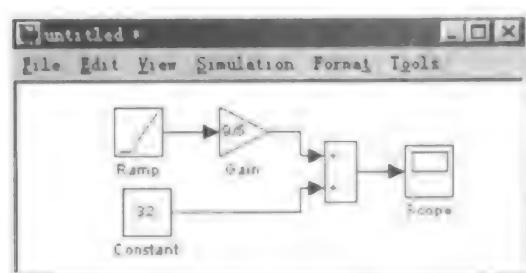


图 4-14 连线好的模型

这样, 将摄氏温度转化为华氏温度的模型就建好了, 读者可以运行仿真, 来观看 Scope 模块的输出。

## 2. 对简单的连续系统进行建模

下面来演示如何对微分方程进行建模, 请看如下的方程:

$$x'(t) = -2x(t) + u(t)$$

其中,  $u(t)$  是幅度为 1, 频率为 1rad/s 的方波信号。积分模块将  $x(t)$  的微分信号积分来获得  $x(t)$ 。此模型中需要的其他模块包括一个 Gain 模块和一个 Sum 模块。要产生方波信号, 可以使用 Signal Generator 模块, 请选择波形为方波 (Square) 并改变频率单位为 rad/sec。同样, 用 Scope 模块来观看最后的输出结果。

把所需的模块复制好后, 请把 Gain 模块的增益参数设为-2。然后按图 4-15 所示, 进行连线。

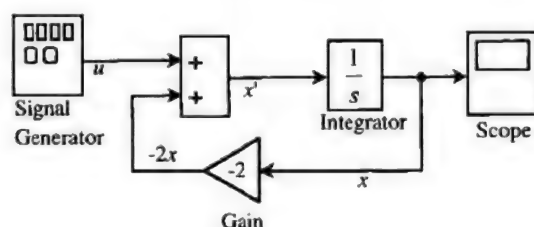


图 4-15 模型图表

图表中, Gain 模块的翻转, 可以通过 Format 菜单下的 Flip Block 命令来进行。

这个模型的一个重要特点是包含了一个由 Sum 模块、Integrator 模块和 Gain 模块组成的环路。在这个方程里,  $x$  是 Integrator 模块的输出, 它同样是计算  $x'$  的模块的输入。这个关系就是通过模型中的环路来实现。运行仿真后, Scope 模块显示的波形如图 4-16 所示。

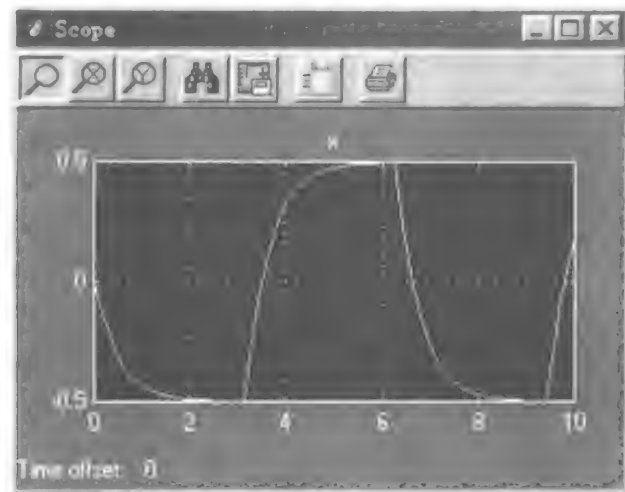


图 4-16 输出结果

本例所处理的同样可以表示为传递函数的形式。很显然上面的微分方程左右取拉氏变换, 即

$$x'(t) = -2x(t) + u(t) \text{ 变为}$$

$$sX(s) = -2X(s) + U(s)$$

$$\Rightarrow \frac{X(s)}{U(s)} = \frac{1}{s+2}$$

这样就不难推出由方程描述的输入  $u(t)$ , 输出  $x(t)$  的系统的传递函数为

$$Fcn(s) = \frac{1}{s+2}$$

于是, 就可以用 Transfer Fcn 模块来进行建模 (Continuous 库)。为此, 要先指定对应的传递函数, 可以在模块参数对话框设置 Numerator (分子) 参数、Denominator (分母) 参数分别为 [1] 和 [1 2]。也就是说, 这个模块是用传递函数的分式形式表示, 且系数从左至右按  $s$  的降幂排列。于是, 上面的模型就可以简化为图 4-17。

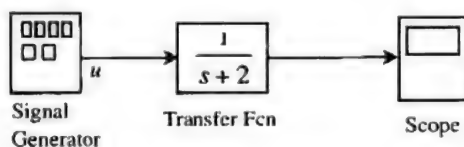


图 4-17 简化后的模型

### 4.3 Simulink 里的数据类型

众所周知，数据类型决定了分配给一个数据的存储资源，决定数据表示的精确性、动态范围、性能和存储资源利用。和 MATLAB 一样，Simulink 也允许用户说明 Simulink 模型中信号和模块参数的数据类型。

这种能指定模型的信号和模块参数的数据类型的能力，在实时控制应用中非常有用。例如，它允许一个 Simulink 模型指定一个最优的数据类型在由代码生成工具从模型生成的代码里表示信号和模块参数。这种代码生成工具比如第十章将讲述的 Real-TimeWorkshop，通过选择合适的数据类型，用户可以大大增强代码的性能，而又缩小代码的大小。

Simulink 在开始仿真之前以及仿真过程中，会进行一个额外的检查，以确认运行的模型的类型安全性。所谓模型的类型安全性，指从该模型产生的代码不会出现上溢或下溢，并且因此产生不精确的结果。使用 Simulink 缺省的数据类型（double）的模型都是固有类型安全的。如果读者可以确信自己不会建立非缺省数据类型的模块，那么这一节对读者的意义不大。话也说回来，读者最好还是能仔细阅读这一节。

#### 4.3.1 Simulink 支持的数据类型

Simulink 支持所有的 MATLAB 内置数据类型，内置数据类型指 MATLAB 自己定义的数据类型，以区别于用户自己定义的数据类型。除了特别说明，本书中的数据类型都是指内置数据类型。表 4-3 列出了所有的 MATLAB 内置数据类型。

表 4-3 MATLAB 内置数据类型

模 块 名	说 明
Double	双精度浮点类型
Single	单精度浮点类型
int8	有符号 8 位整数
uint8	无符号 8 位整数
int16	有符号 16 位整数
uint16	无符号 16 位整数
int32	有符号 32 位整数
uint32	无符号 32 位整数

除了内置数据类型，Simulink 还定义了布尔类型，取值为 0 或 1，它们的内部表示是 uint8（无符号 8 位整数）。

所有的 Simulink 模块都缺省地接受 double 类型的信号，但有些模块需要布尔类型的输入，而另外一些模块支持多数据类型输入，还有些则支持复数信号。表 4-4 列出了这些模块。

表 4-4 Simulink 里支持复数的模块

模 块 名	说 明
Abs	输入或者输出具有 double 类型的实数或者复数信号，输出 double 类型的实数
Combinatorial Logic	输入和输出的类型，在布尔模式使能时只能为布尔类型，否则可以为 double
Constant	输出具有任何类型的实数或复数信号
Data Type Conversion	输入、输出具有任何类型的实数或复数信号
Demux	接收混合类型的信号向量
Display	接收具有任何复数或者实数数据类型的信号
Dot Product	输入和输出 double 类型的实数或者复数信号
Enable	相应的子系统使能端接收布尔或者 double 类型
From	输出连接到对应 Goto 模块的信号的数据类型
From Workspace	输出类型为对应的工作空间的数据值的类型
Gain	输入可以是任何数据类型的实数或者复数信号和向量
Goto	输入可以使人任何数据类型
Ground	输出与之相连信号相同类型的 0 信号
Hit Crossing	输入 double 类型的信号，输出布尔类型
Inport	可以接收任何数据类型的信号，当为系统层的输入端或者是直接和同子系统的输出端直接相连时，那么它的各个元素信号的数据类型必须相同
Integrator	该模块在它的数据端口接收 double 类型，在置位端接收 double 和布尔类型
Logical Operator	输入和输出布尔类型的实信号，不在布尔模式还可处理 double 类型
Manual Switch	接收任何类型的实数或者复数信号，所有输入端信号的类型必须相同
Math Function	输入和输出 double 类型的信号

关于模块的输入、输出信号支持的数据类型的详细说明，请见本书的附录。一个模块如果没有说明它支持的数据类型，那就表示它只能支持 double 类型的数据。

在设置模块参数时，指定一个值为特殊数据类型表示的方法为：`type (value)`。例如要把常数模块的参数设为 1.0 单精度表示，则可以在参数对话框输入。

```
>> single (1.0);
```

而把一个具有特定数据类型的信号引入模型的方法有：

(1) 通过根目录 `inport` 或者 `FromWorkspace` 模块将 MATLAB 工作空间中具有指定数据类型的数据信号导入你的模型；

(2) 建立一个常数模块并设置它的参数为指定的类型；

(3) 使用 `DataTypeConversion` 模块将一个信号转换为指定的信号。

若要显示一个信号的数据类型，可以从 `format` 命令选择 `PortDataTypes` 命令，对于这一命令的用法，作者在前一章提过，这里也不赘述。

### 4.3.2 数据类型传播

无论用户何时开始仿真、显示端口数据类型和更新数据类型显示，Simulink 都会进行一个称为数据类型传播的处理步骤。这个步骤确定没有被特别设定的信号的数据类型，以及检查信号的数据类型和输入端口是否冲突。如果冲突，Simulink 会显示一个提示对话框，告诉用户出现冲突的信号和端口，它还会把冲突信号的路径用亮条加以显示。读者可以在模型里插入一个 Data type conversion 模块来解决类型冲突。

下面的几条法则将有助于读者建立类型安全、运行不出错的模型。

(1) 信号的数据类型一般情况下不会影响参数的数据类型。但有一个重要的例外，那就是常数模块，因为它的输出值决定于所设置的参数，所以输出端口的数据类型就决定于所设置的参数。

(2) 如果模块输出端的信号是输入信号和模块参数的函数，并且输入信号和参数的数据类型不一致，Simulink 在计算模块输出之前会自动的把参数的数据类型转化成输入类型。

(3) 总的说来，输出的数据类型决定于模块的输入数据类型，除了两个例外：Constant 模块和 Data type conversion 模块，它们的输出类型是由参数的类型决定的。

(4) 虚拟模块接收任何数据类型的信号作为输入。

(5) 连接到非虚拟模块同一端口的向量信号的元素信号必须具有相同的数据类型。

(6) 连接到非虚拟模块输入端口的所有信号必须具有相同的类型。注意和第(5)点的区别，这里是指信号之间的关系，而第(5)点强调了同一信号的不同元素的数据类型。

(7) 控制端口（例如，Enable 模块和 Trigger 模块，请见本章的条件子系统一节）接收布尔类型或 double 类型的信号。

(8) Solver 模块只接收 double 类型的信号。

(9) 禁止过零检测的模块只能和非 double 数据类型的信号相连接。

缺省情况下，当 Simulink 检测到接收 double 类型数据的模块和只选择布尔类型的模块相连时，不会给出错误提示，这主要是为了和以前版本的模型兼容。用户可以通过在 Simulation Parameters 对话框的 Diagnostics 页，去除 Relax boolean type checking 选项的选中状态，来使 Simulink 实施严格的布尔类型检验。

前面曾讲过，在仿真过程中，如果输出信号是模块输入和参数的函数，计算这种输出信号时，Simulink 会把参数类型转换到信号的数据类型。对于这条法则，有下面的几个特例：

(1) 如果信号数据类型无法表示参数值，Simulink 将中断仿真，并给出错误信号。

请看下面的模型（图 4-18），表 4-5 列出了模型中各模块的参数值。

表 4-5 模型所需模块

模块名称	参数设置
Constant	Constant:uint8 (1)
Gain	Gain: int32 (255)

这个模型用 Gain 模块将 Constant 模块的输入信号放大，计算 Gain 模块时需要计算输入信号和增益的乘积。这就要求两个值的类型相同。然而，这个模型中 Constant 模块的值是无

符号类型，而 Gain 模块的值却是 int32 类型的。于是，Simulink 就要进行一个类型转换，把参数的类型映射到信号类型。如果信号的数据类型能够表示参数的数据类型，Simulink 就完成这种转换。因为 Gain 的值是 255，恰巧在无符号数的表示范围（0-255）内，所以 Simulink 可以完成这种转换，仿真运行不会产生错误。

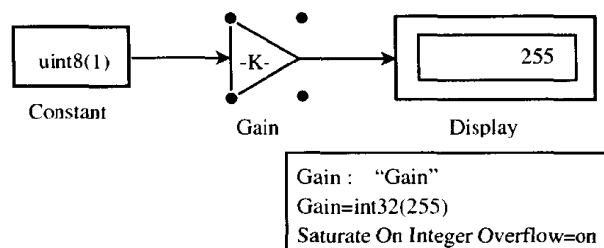


图 4-18 示例模型

但是如果把 Gain 模块的参数 Gain 改成 256，那么就超出了无符号数的表示范围了，Simulink 无法进行转换，它会给出一个错误信号。

(2) 如果信号的数据类型能够表示参数的值，仅仅是表示损失的精度，Simulink 会继续仿真，并在 MATLAB 命令窗口给出一个警告信息。例如把上例中的 Gain 参数设为是 double (25.4)，由于它处在无符号数表示范围内（0-255），Simulink 就会把 25.4 的整数部分截取，并转换成无符号数，所以最后的结果是 25，但给出了一个精度损失警告。但如果是 25.0 就不存在精度损失，所以也不会有警告信息。

- ✎ 从 int32 到 float 或 double 的数据类型转换，有可能会产生精度损失，当参数的量级很大时，这种损失将会很严重。如果一个 int32 类型的参数转换产生精度损失，Simulink 会给出警告信息。

### 4.3.3 在模型里使用复数信号

缺省情况下，Simulink 的信号的值都是实数，然而，模型也可以建立和处理值为复数的信号。

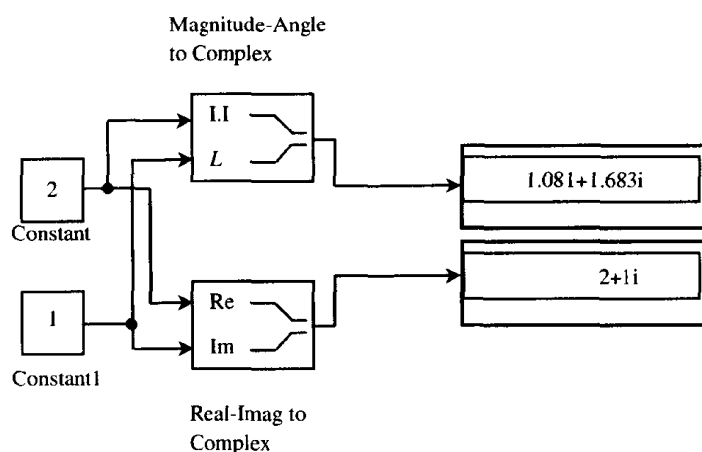


图 4-19 产生复数信号的示例模型

有下面几种方法可以把复数信号引入到模型中：

- (1) 把 MATLAB 工作空间里的复数信号通过 root-level import 导入到模型里;
- (2) 复制一个 Constant 模块到你的模型, 并把它的参数设为复数;
- (3) 分别产生对应着复数信号实部和虚部的两个实信号, 然后把它们联合成一个复数信号。这里要用到 Real-Imag to Complex 模块, 它的位置在 Simulink 库的 math 子库, 在 math 子库里还有一个 Magnitue-Angle to Complex 模块, 它是把两个输入作为辐值和辐角来生成复数。

图 4-19 给出了一个演示如何产生复数信号的示例模型, 读者可以看到使用这两个模块得到的结果完全不同。此外, math 子库还提供了一个把复数分解为实部和虚部的模块——Complex to Real-image 模块。

## 4.4 建立子系统

### 4.4.1 建立子系统

当模型规模很大、很复杂时, 可以通过把一些模块组合成一个子系统, 来简化模型。建立子系统有以下几个优点:

- (1) 可以减少显示在模型窗口的模块数, 这样用户的模型窗口就会很整齐, 也方便用户连线;
- (2) 可以将功能相关的模块放在一起, 用户可以用建立子系统创建自己的库模块;
- (3) 可以生成层次化的模型图表, 即子系统在一层, 组成子系统的模块在另一层。这样用户在设计模型时, 既可采用自上而下的设计方法, 也可以采用自下而上的设计方法。

在 Simulink 里创建子系统的途径有两种:

- (1) 增加一个子系统模块到你的模型, 然后打开这个模块并在打开的模型窗口建立子系统;
- (2) 先把模块链接好, 然后再把这些模块组合成子系统。

#### 1. 通过子系统模块来建立子系统

这种方法首先往模型里加入一个称为 Subsystem 的库模块, 然后再往该模块里加入组成子系统的模块, 进行设计。

下面建立一个简单的子系统作为示例。该系统的功能是将输入的华氏温度转换为摄氏温度, 读者可以在库浏览器的 Simulink extras 节点的 Transformation 库找到它的这个模块的原始版本。图 4-20 是建好的子系统。

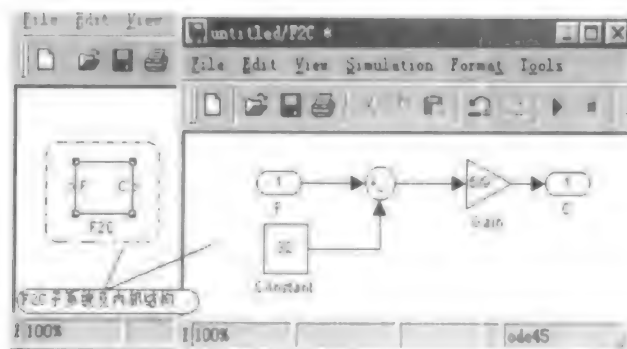


图 4-20 F2C 子系统及其内部结构

它的建立可以按下面的步骤进行:

(1) 将 subsystem 模块复制到模型窗口, 它的位置在 Simulink 结点的 signal&system 子库里。

(2) 双击 subsystem 模块, 这会打开子系统的编辑窗口。见上图右面的窗口。

(3) 将组成子系统的模块添加到子系统窗口。为了定义子系统的输入、输出端口, 必须手动地向子系统窗口添加 in1 和 out1 模块。这是两个虚模块, 与它们相连的信号将作为子系统的输入输出信号。当然这只是 Simulink 图形化建模的一个逻辑方法, 事实上在物理模型里这是没有必要的, 但在 Simulink 建立子系统里它们是不可或缺的。

(4) 按设计好的图表, 连接好各个模块, 并修改 in1 和 out1 模块下面的标签。实际上, 所谓标签就是这两个端口在子系统图标上的显示文字, 如上图中的 F 和 C。

(5) 关闭子系统窗口, 把模型保存。读者可能想直接用子系统窗口 File 菜单里的 save 命令来保存子系统, 但这个想法是行不通的。其实, 这个命令也是保存整个模型, 因为在 Simulink 里只有模型或库才是一个完整的独立的实体。此外, 重新定义子系统的名称是一个很好的习惯。

至此, 就建立了一个子系统。

## 2. 组合已存在的模块来建立子系统

如果现有的模型里已经包含了用户想转化成 subsystem 的模块, 就可以通过组合这些模块的方式来建立子系统。

例如, 现在已经有有了一个如图 4-21 所示的模型。这个模型的功能是把离散信号累加。读者可以运行这个模型, to workspace 的功能是把存储仿真结果的输出变量保存到 MATLAB 的工作空间里。所以, 要查看仿真结果, 只需在 MATLAB 命令窗口查看这个变量, 这里是 simout。注意, 这个变量是结构类型, 结构内的字段的存取在上一节讲过, 不清楚的读者可以再回过头复习一下。更详细的信息可以在 MATLAB 的帮助文档里找到。

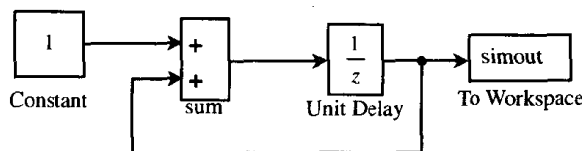


图 4-21 根据已有模型建立子系统

读者会看到, sum 模块和 unit delay 模块组合在一起完成了一个累加器的作用。下面将演示如何把它们转化为子系统。

首先请读者按照图 4-21, 建立好这个模型。其中, unit delay 模块和 to workspace 模块的位置分别是 Simulink 节点下的 discrete 和 sink 子库。

把已存在的模块转化为子系统, 只要用 Simulink 用户界面 Edit 菜单下的 Create Subsystem 命令即可。但读者会发现此时的该命令处于不可用状态, 这是因为你还没有选择合适的操作对象。所以第一步操作应该是选取要组合成一个子系统的模块, 这里别忘了连线也是对象之一。前面曾讲过如何在 Simulink 里选择多个对象, 据笔者的经验, 第一种逐个选取的方法在这里不能起作用, 正确的方法是用方框包含待选择对象。正确选择好操作对象之后, Edit 下的 Create Subsystem 命令就会变成可用状态。单击它之后, Simulink 会自动将操作对象转化



成一个子系统，转化后的图表如图 4-22 所示。

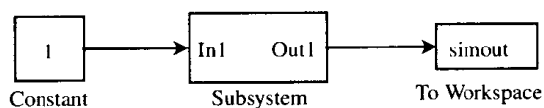


图 4-22 自动生成的子系统

读者双击生成的子系统打开子系统窗口，将会发现自动生成的子系统其内部结构和前一手动方法生成的图表结构是一致的。也就是说两种方法的区别只在于输入输出端口的加入是自动还是手动。

这种方法生成的子系统美中不足的是端口名不直观，需要用户打开子系统窗口手动修改端口模块下的标签。

- ✎ 模型里的模块标签必须是唯一的，因为 Simulink 把它作为此模块的名称，来标识模块，这一点在学习了模型文件的格式（第五章）之后将会看得很清楚。

#### 4.4.2 用子系统来自定义库

在 4.1.2 节曾提到过如何建立自己的库，但是那里所建立的库有一点不好，就是不能在库浏览器里浏览。这一小节将介绍如何往库浏览器里添加节点。

例如，把前面生成的 F2C 子系统添加到 Simulink extras 节点的 transformation 子库里。读者稍微思考一下，就能找到这种方法。假定 F2C 子系统已经建好，建立使用的方法上面讲到的两种都可。首先，打开 Transformations 子库，方法是用右键弹出的菜单的“open...”命令。然后用 unlock library 命令去除锁定，使该子库处在可修改状态，再把已经建好的子系统 F2C 复制到 Transformations 子库窗口。操作还没有完全结束，因为用户所建的子系统里的模块有可能包含参考模块，也就是说它们不是独立的。而且，即使子系统内不包括参考模块，但复制在 Transformations 子库里的子系统其实是你前面建好的子系统的参考模块，也就是原来的子系统才是真正的库模块。这种情况的后果是当你存放初始子系统的模型或库有所变动时，存放在 Transformations 子库里的子系统就变成链接的模块，这显然不符合库模块的独立存在性要求。基于上述原因，一定要把复制在子库里的子系统的链接断掉，这一点可以用 Edit 菜单下的 break library link 实现。最后保存改动后的库，就完成了添加。图 4-23 反映了添加后的效果。

读者已经注意到，添加进去的 F2C 在形式上更像是一个子库，而不像是一个模块。这是因为添加进去的本身就是一个子系统，而非一个模块，子系统的内部图表是可以浏览的。可以用封装子系统的方法，使它表现得像一个模块。关于封装子系统这个主题的详细信息将在下一节介绍，这里只对 F2C 进行简单的封装，用以演示封装后添加到库浏览器的效果。

封装的方法很简单，请选择好初始建立的子系统（笔者不赞成读者直接在子库窗口上操作），从 Edit 菜单选择 mask the subsystem 命令，这个命令也可以在右键弹出的菜单里找到。这将打开一个 mask edit 的对话框，请选择 document 页，在 blocks description 编辑框里输入一行文字，比如“这是一个把华氏温度转换成摄氏温度的自定义模块”，按 OK 按钮关闭对话框。再按照前面讲过的步骤把它添加到 Transformations 子库。图 4-24 是封装后相应的情形。

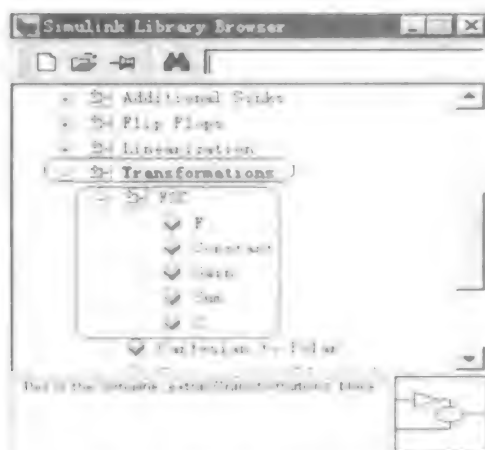


图 4-23 添加 ICC 模块到库浏览器



图 4-24 封装好的子模块

这时，封装后的子系统已经表现得和库模块毫无二致了。

从图 4-23 可以得到另外一个启示，即库浏览器里，子库并不是一个库文件，确切地说是子系统。库文件由子系统构成，子系统里存放下一级子系统或者模块。于是用户完全可以生成自己的子库，添加到某一个库文件里，这样就可以在库浏览器里显示该子库了。用户自定义的子库通常放在 Simulink extras 库里，例如，可以把常用的模块放在一起组成一个“我的模块”子库（见图 4-25）。明白子库即子系统的道理，读者不难完成这种添加，记住两点，一是要把子系统内的参考模块的链接去掉，二是不要封装子系统。



图 4-25 添加的“我的库”

## 4.5 封装子系统

在前面一节里，我们曾略微提到封装一个子系统可以使子系统表现得和模块一样——双击后会出现一个参数设置对话框，这反映了封装子系统好处的一个方面。

封装是 Simulink 的一个十分重要的特性，它使得用户可以自定义子系统的对话框和图标。通过封装，用户能够：

(1) 用一个子系统一个参数设置对话框代替多个对话框，以简化模型的使用。否则，使用模型时，就不得不逐个打开子系统内的各个模块的参数设置对话框输入参数。经过封装之后，这些参数就可以通过封装好的子系统对话框来设置，并传给封装过的子系统内的模块。

(2) 定义一个具有用户自己的模块描述、参数字段标签和帮助文档的对话框，为模型使用者提供一个描述性更强，更友好的用户界面。

(3) 定义命令，计算那些取值依赖于模块参数的变量的值。

(4) 定义一个能反映子系统目的，更有意义的模块图标。

(5) 把子系统的内容隐藏在定义好的界面下，避免使用者对子系统进行无法预料的改动。

(6) 建立动态对话框。

下面就来介绍如何对子系统进行封装。

### 4.5.1 子系统封装示例

图 4-26 是一个用以计算  $y=mx+b$  的简单子系统，其中的右图是该子系统的内部结构。它内部的模块有：一个增益参数为  $m$  的 gain 模块，一个常数值为  $b$  的 Constant 模块，以及一个 sum 模块。在进入下面的讨论之前，请读者先把这个子系统建好。

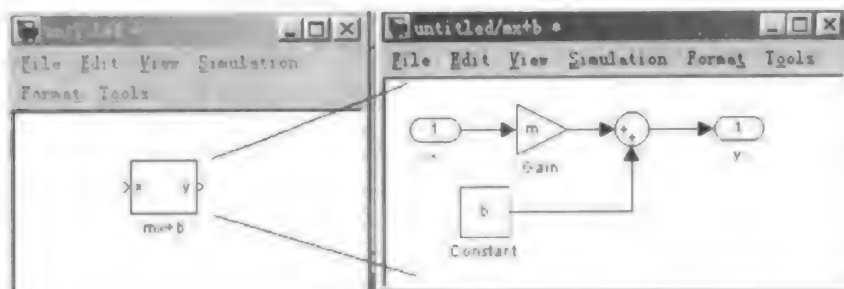


图 4-26 封装好的子系统

如果不进行封装，正如以前看到的，双击子系统会看到子系统的内部结构。为了改变  $m$  和  $b$  这两个参数，你必须分别打开 Gain 和 Constant 的参数对话框逐个设置，如果子系统内的模块很多，那么这项工作将是十分烦人的。

但是，封装可以简化这些操作。按照惯例，在开始讲解如何封装之前，本书先让读者体会一下子系统封装后的不同凡响之处。图 4-27 就是双击封装过的子系统弹出的对话框。

此时，设置  $m$  和  $b$  的值只需打开一个对话框就可以了。

下面，就让我们来看看在 Simulink 里是如何封装子系统的。大体上说，一个封装要完成

## 3 件事:

- (1) 定义提示对话框及其特性;
- (2) 定义被封装的子系统的描述和帮助文档;
- (3) 定义产生模块图标的命令。

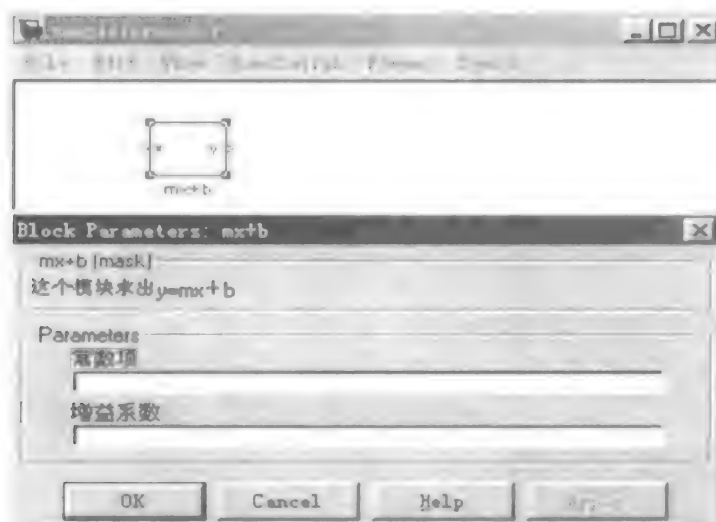


图 4-27 封装过的子系统的参数设置对话框

## 1. 产生提示对话框

封装一个子系统，首先要选择该子系统，然后再从用户界面的 **Edit** 菜单选择 **Mask Subsystem** 命令。于是就会出现读者前面已经见过的封装编辑器，它分为三页，出现在最前面的通常是 **Initialization** 页。图 4-28 是本例中该页呈现的样子。

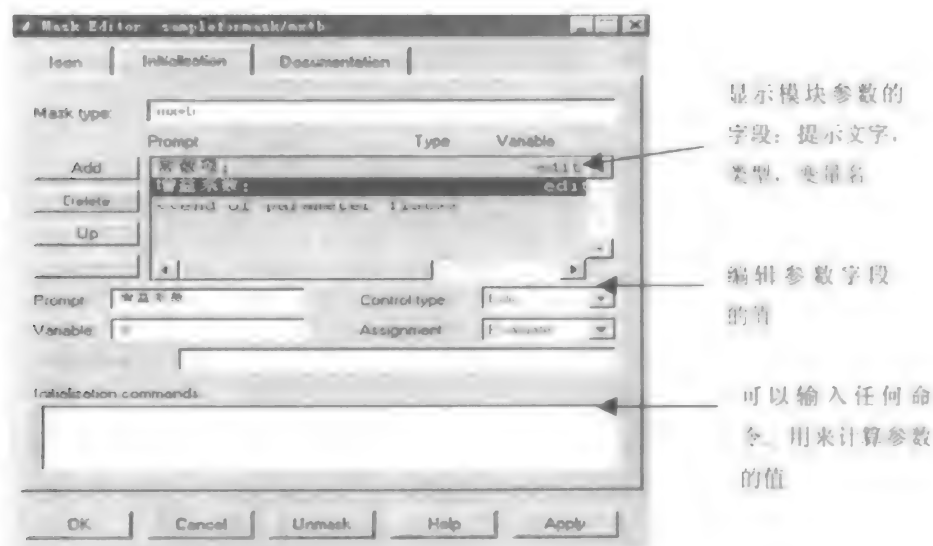


图 4-28 封装编辑器的 Initialization

正如在图上所看到的一样，用户可以定义的封装参数的属性有：**prompt**、**type** 和 **variable**。其中，**prompt** 是指用以描述参数作用的提示性文字。**type** 是指用于输入或选择参数的控件的

样式，记住这里不是指参数的数据类型，比如在本例中选择了 `edit` 控件。`Variable` 指定存储参数值的变量名，也就是说用户在参数对话框输入的参数值将自动传给这些变量。实际上，这些变量存在于模块的封装工作空间——它是 `Simulink` 专门为模块分配的工作空间。它是用户定义变量的场所，只有它里面的变量才能被模块访问。但是，并不是所有的变量都只能是参数，只有像 `b` 和 `m` 这样在对话框里定义的变量才是需要用户设置的参数，实际上子系统里有很多模块的参数值都可以被预先设置好。读者可以从 `Simulink` 的许多 `demo` 里发现这样的现象，许多模块的参数设定为一个变量，但是找遍了整个模型，就是找不到给该变量赋值的地方，而模型却能运行正确。其中的奥妙在于使用了回调函数，本书的后面章节将会为读者讲解这一技巧。以上是题外话，下面还是接着封装的主题。

将这些属性值设置好了后，按 `OK` 键，再重新双击子系统，就可以得到前面图 4-27 所示的效果。

## 2. 建立模块的描述信息以及帮助文档

模块的描述信息和帮助文档是在封装编辑器的 `documentation` 页设置的。这一页的结构很简单，只有几个属性值，它们各自的功能读者很容易就能实验出来。图 4-29 描绘了它们与参数设置对话框里几个要素的对应关系。



图 4-29 描述信息和帮助文档的显示位置

## 3. 生成模块的图标

到目前为止，我们已经为 `mx+b` 定义好了一个自定义的参数设置对话框。然而，子系统模块依旧显示着模块的通用图标，这对使用者而言是很不直观的。一个比较好的替代图标是用一条斜线来表示。

模块图标可以在 `icon` 页定义。

请在 `drawing commands` 提示下的 `Edit` 里输入命令：`>> plot ([0,1],[0, m]+ (m<0) ) %` 这里“`>>`”依然表示是 `MATLAB` 命令。

上面用作图命令画一条从点  $(0,0)$  到点  $(0,m)$  的直线，当  $m<0$  时，就将直线平移 1 以保证所画直线在模块框区域内。而且读者还可能注意到了在上述命令中，用到了变量 `m`，事实上，作图命令可以访问 `mask` 工作空间里的任何变量。这样随着参数 `m` 的改变，模块的图标也会相应的改变，这就是一种动态图标。因为在用户还没有设置参数值时，`m` 依旧是个不确定量，那画图命令就无法运行了。

此外，还要把 `icon` 页上的 `drawing coordinate` 属性设置为 `normalized`。这样作图区域就被限制在左下角为  $(0,0)$ ，右上角为  $(1,1)$  的正方形区域里。至于该页上的其他属性均不用修

改，使用原来的缺省值，它们的含义在后面会具体讲到。

这样，再按 OK 按钮关闭封装编辑器就可以了。图 4-30 是封装后的图标样子。

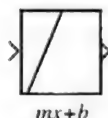


图 4-30 建立的图标样子

至此，我们封装示例就完成了。下面的一小节，将更加深入细致地讲解封装编辑器。

#### 4.5.2 initialization 页

在前面的一小节里，这一页的大部分操作都已经讲过，所以在这就不再赘述。这里的注意力将会集中在前面有意回避了的一些内容。

##### 1. assignment 属性

assignment 属性的类型有两个值可以选择：evaluate 和 literal。如果 assignment 属性设置为 evaluate，那么模块使用者在模块参数设置对话框里输入的值，在被赋给变量之前，首先由 MATLAB 进行估值。如果 assignment 属性设置为 literal，则输入的值，不经过估值，而是直接作为字符串传给变量。为了弄清这两者的区别，请读者按图 4-31 建立一个测试模型。

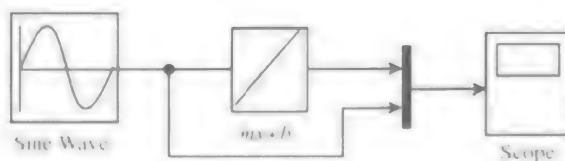


图 4-31 测试模型

实验的思路是：分别将 assignment 属性设置成 evaluate 和 literal，然后再输入相同的字符串，用以对比两者的不同效果。

首先，用用户界面的 edit 菜单下的 edit mask 命令对  $mx+b$  模块重新封装，这里只要把 assignment 属性设置为 evaluate。按 OK 按钮关闭后，再设置  $m$  和  $b$  的值，不妨设置  $b$  为 0，而在  $m$  的编辑框里输入字符串：gain（见图 4-32）。



图 4-32 参数的设置

关闭参数对话框之后，运行仿真模型。这时，会出现一个错误提示（图 4-33）。为了解

决这个问题，请回到 MATLAB 命令窗口，输入

```
>> gain = 2;
```

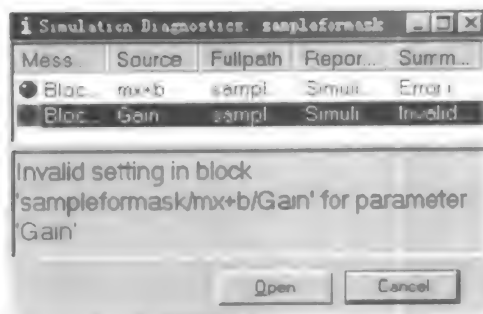


图 4-33 错误提示对话框

然后再运行仿真，这时就不再出现错误提示了，而且观察仿真的结果会发现，输出恰好是输入的两倍，也就是说， $m$  的值被设成了 2，恰恰是我们在 MATLAB 工作空间里为  $gain$  赋的值。这是怎么回事呢？

很简单，前面讲过，在对话框输入了  $gain$  之后，一旦开始运行仿真，就先由 MATLAB 进行估值，这就相当于在 MATLAB 命令窗口输入了

```
>> gain
```

按照以前所讲的，MATLAB 对输入的变量估计顺序，首先它将检查是不是 MATLAB 工作空间里的变量，如果不是，再看是不是内置函数，如此下去，具体的顺序读者请看第二章。所以在 MATLAB 工作空间里定义  $gain$  变量之前，由于它不是内置函数，也不是搜索路径中的 mex 文件或 m 文件，MATLAB 自然无法进行估值，于是就会出错。但定义了  $gain$  变量之后，情况就不同了，MATLAB 把  $gain$  的值传给参数变量  $m$ ，所以出现前面的仿真结果是很自然的结果，并不是偶合。

以此类推，在  $m$  的编辑框里输入一个函数名，也是可以的。读者不妨自己试试，例如  $abs(-1)$  等等。

这些概念，在学习过如何编写 S 函数之后，将是很显然的事情。

那 literal 又是怎么回事呢？

为此请重新打开  $mx+b$  的封装编辑器，把参数  $m$  的 assignment 属性设置成 literal，别的属性的设置保持不变，然后重新在模块的参数设置对话框设置  $m$  的值为  $abs(-2)$ 。

```
>> abs(-2); %在编辑框里输入 abs(-2)
```

输入完后，请运行仿真模型。结果是可以预想的，Simulink 会给出一个错误提示。双击这些 Simulink 里的错误提示，Simulink 会告诉你错误在模型中的哪个位置，读者可以自己试试这些功能。

我们来分析出现错误的原因。因为封装中，参数  $m$  的 assignment 已经被设置为 literal 了，于是在参数设置对话框输入的 ' $abs(-2);$ '，MATLAB 并不对它进行估值，直接当成字符串赋给参数变量  $m$ ，让模块自己去处理。于是  $m$  的值就是 ' $abs(-2);$ '，当子系统里的  $gain$



模块访问  $m$  时得到的值就是 ‘abs (-2);’, 自然会出错了。改正的方法很简单, 基本原理是由用户自己进行估值。为此, 请读者再打开封装编辑器, 在 initialization commands 里输入

```
>> m=eval (m);
```

eval 命令是用来对字符串进行估值的, 更确切地说, eval 命令把输入的字符串, 作为命令来执行。比如上面的  $m$  的值是 ‘abs (m);’, 当使用 eval 命令时, 它就把字符串的内容 abs (m) 当成命令执行, 然后返回执行后的结果, 也即  $m=\text{eval}('abs (-2);')$ , 实际上就是  $m = \text{abs} (-2)$ 。

如此改动之后, 再执行模型就不会出现错误了。

估值的位置也不一定非要出现在初始命令, 也可以是在子系统内部模块访问  $m$  变量时, 即在 gain 模块的参数编辑框里输入

```
>> eval (m);
```

这样的效果是一样的。

一般, 在实际使用时, literal 并不常用, 通常使用的是 evaluate。

## 2. 控件类型

通过选择控件类型, 用户可以决定是以直接输入的方式还是以选择输入的方式输入参数值。Simulink 提供了三种控件类型: editfields、checkboxes 和 popupcontrlds。图 4-34 显示了一个使用了三种控件类型的参数设置对话框。

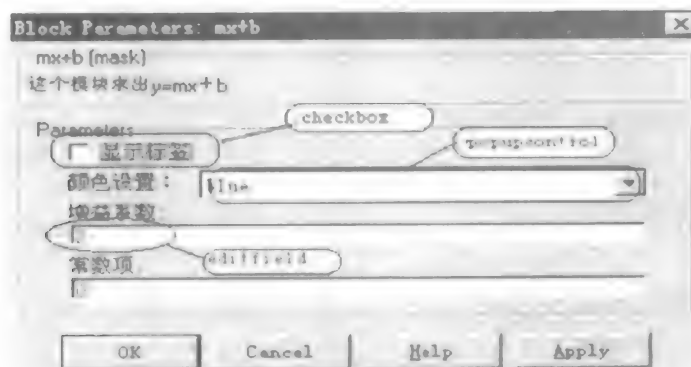


图 4-34 三种控件类型示意图

其中, editfield 控件是前面的例子中使用过的, 也是最常用的一种。设置参数时, 用户必须直接在编辑框里输入要设置的值, 正如前面所看到的一样。

在这里, 要对前面关于 assignment 属性的一些论述加以修正。应该说前面所讲的原则大体上是没错的, 也就是估值和不估值的区别。但前面的论述只是在控件类型是 editfield 时才对。表 4-6 描述了在此情况下, 参数是在不同的 assignment 属性下的赋值方式。

表 4-6 edit field 的变量取值与 assignment 属性的关系

Assignment	取 值
Evaluate	参数变量取值为输入表达式进行估值后获得的结果
Literal	参数变量取值为输入表达式的实际字符串



与 editfield 的直接输入不同的是，checkbox 只能让用户在选择 checkbox 和不选择 checkbox 两种备选项里选择。定义一个 checkbox 也没有什么特别之处，和 editfield 类似。但和 checkbox 变量关联的变量的赋值方式不一样，它如表 4-7 显示。

表 4-7 checkbox 的变量取值与 assignment 属性的关系

Checkbox	Evaluate 方式下参数变量的取值	Literal 方式下参数变量的取值
选中	1	'on'
未选中	0	'off'

和前面两种方式相比，popupcontrol 略微有些不同。它的作用也是提供备选项让用户选择，但它不像 checkbox 那样只有两个，它提供了所有可能的取值的列表。一旦 control type 选择了 popupcontrol，那原来一直处于不可用状态的 popupstrings 编辑框将变亮，它是用来输入 popup 列表里的所有可能取值项。不同的子项要用“|”来隔开，图 4-35 是它的示意图。

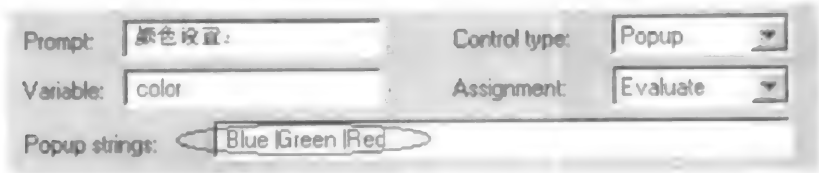


图 4-35 定义 popup strings

表 4-8 则是与 popup control 关联的变量取值与 assignment 属性的关系。

表 4-8 popup control 中变量取值与 assignment 属性的关系

Assignment	取 值
Evaluate	选项的索引值，索引值从 1 开始排列，例如，当第三个选项被选择，参数值为 3
Literal	标识选项的字符串，例如第三个选项被选择，参数值就为 'Red'

3. 可调参数 (Tunable Parameterst)

所谓的可调参数是指能够在运行时动态修改的封装参数。当你建立一个封装，它的所有参数都是可调的。但用户可以通过模块的 MaskTunableValues 属性，来控制封装参数的可调与否。用户也可以为封装过的库模块定义缺省参数值，操作的具体步骤如下：

- (1) 解除库锁定状态。
- (2) 打开模块的参数对话框，在编辑框输入要设定的缺省值，然后关闭对话框。
- (3) 保存模型。

于是当这个模块复制到模型中时，对话框打开，缺省参数值将会出现在模块的对话框。

4. 初始化命令 (Initialization Commands) 及其调试

初始化命令用于定义驻留在模块的封装工作空间内的变量，这些变量通常不希望模块使用者修改它，但它又是必须的。在初始化命令里定义的变量能被子系统内的模块访问，也能被其他的初始化命令访问，还能被制作图标命令访问。总之，和封装的参数变量没什么区别。

初始化命令在以下几种时候被执行：

- (1) 模型被导入。
- (2) 开始仿真，或者是模型的图表被更新（指调用 `update datagram` 菜单命令）。
- (3) 被封装的模块被旋转。
- (4) 模块的图标被重画并且画图命令用到在 `initialization commands` 定义的变量。

初始化命令，可以是任何的 MATLAB 有效表达式，包括 MATLAB 函数、运算符以及在封装工作空间里定义过的变量。

初始化命令的调试，一般有三种方法：

- (1) 把每个命令后的分号去掉，使命令的执行结果能在 MATLAB 命令窗口显示。
- (2) 放置一个 `keyboard` 命令到初始化命令之中，使得运行能被键盘操作中断，并把控制权交给键盘。关于 `keyboard` 的详细信息，请查阅帮助。
- (3) 在 MATLAB 命令窗口，输入下面两个命令之一

```
>> dbstop if error
>> dbstop if warning
```

前者的作用是一旦初始化命令产生错误，其运行将被中断，用户就可以检查封装工作空间；后者则是在发生警告时才如此。详细信息，请看 `dbstop` 的帮助。

#### 5. 封装子空间

一旦模块的封装里（注意 Simulink 提供的模块也同样是经过封装得到的，但不一定是子系统），包含初始化命令或者定义了参数变量，Simulink 就为模块建立一个局部的封装工作空间。

被封装的模块不能访问 MATLAB 的工作空间或者其他的封装工作空间，这一点就像前面讲过的 M 文件函数的局部工作空间一样。封装工作空间的内容包括封装参数和由初始化命令定义的变量，这些变量能被封装的模块访问，如果是子系统，那它能被子系统的所有模块访问。

把封装工作空间同 M 文件函数使用的局部工作空间进行类比，有助于加深对前者的理解。读者可以把在内部模块（针对子系统而言）的参数设置对话框里输入的表达式，以及在封装编辑框里输入的初始化命令或者制作图标命令，当成 M 文件函数的一行语句。而子系统的内部模块，就像 M 文件函数内要调用的其他 M 文件函数一样，内部模块的封装空间（如果有的话）和子系统的封装空间是不相同的，它们之间的联系仅仅是内部模块的模块参数。

在前面的  $mx+b$  模块中， $m$  和  $b$  作为定义好的封装参数被存储在  $mx+b$  的封装工作空间里。但它们不会被自动地赋给子系统内的 Gain 模块或者 Constant 模块，而要这些模块来访问这些变量。正如  $mx+b$  模块封装内的命令，如初始化命令，不能使用在 MATLAB 工作空间定义的变量一样，gain 内部的命令也不能直接访问  $mx+b$  封装工作空间里的变量，而只能通过 Gain 模块自身的封装参数，如与 'gain' 提示文字相关的变量，来传入这些值。同样， $mx+b$  的封装工作空间也不能看到 gain 的封装空间定义的变量，这也就是封装的意义。图 4-36 可以大体反映这种关系。

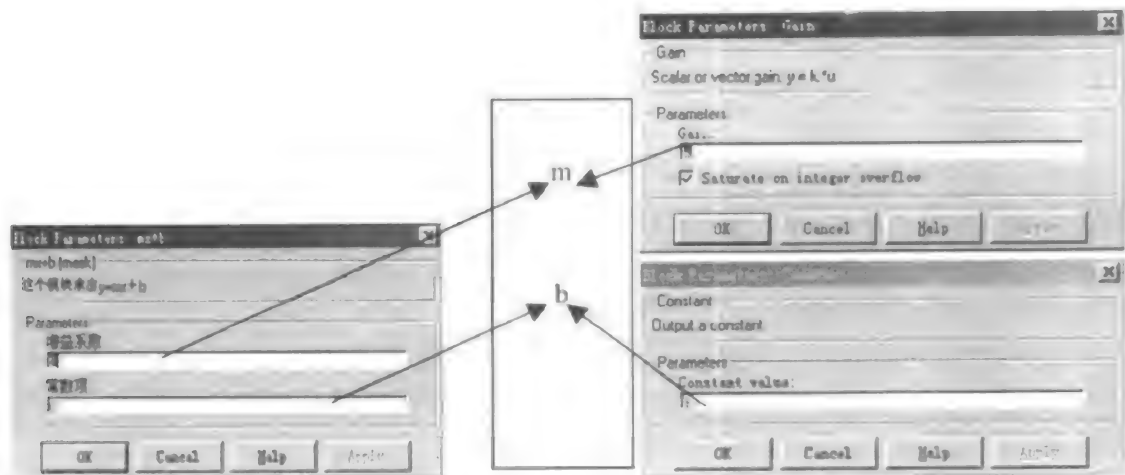


图 4-36 封装子空间

### 4.5.3 icon 页（图标页）

我们知道，icon 页的主要作用是让用户能够自定义模块的图标，它的内容相对简单，只有一个画图命令栏以及几个控制图标属性的 popupcontrols。这一小节将讲述如何产生不同形式的图标，包括：在模块图标上显示文字、在图标上显示图形、在图标上显示图像以及在图标上显示传递函数。最后还会讲解如何控制图标的属性。

#### 1. 在图标上显示文本

定制图标的画图命令都是写在画图命令栏里的，画图命令访问封装工作空间的任何变量，这是前面讲过的内容。要在图标上显示文字，可以使用的命令有：`disp`、`text`、`fprintf` 和 `port_label`。

- ✎ 尽管这些命令在名字上和许多的 MATLAB 函数相同，但它们在功能上却不完全相同。关于这些函数使用在定制图标时的用法，仅限于下面提到的，切不可将 MATLAB 函数的用法移植过来。后面关于定制图标的命令，也都是如此，它们的用法也只有此节中提到的。

它们的用法分别有：

`disp('text')`

`disp(variablename)`

`text(x, y, 'text')`

`text(x, y, stringvariablename)`

`text(x, y, text, 'horizontalAlignment', halign, 'verticalAlignment', valign)`

`fprintf('text')`

`fprintf('format', variablename)`

`port_label(port_type, port_number, label)`

其中，`disp` 命令可以把字符串或变量的内容显示在图标的中心，这里的变量的内容不一定是非要是字符串，也可是数组。

Text 命令的作用是，放置一个字符串（输入参数 `text` 或者 `stringvariablename` 的内容）在由输入参数  $(x, y)$  指定的位置。坐标的单位取决于图标画图坐标属性的设置。从上面的用法列表还可看到：Text 提供了两个参数 '`horizontalAlignment`' 和 '`verticalAlignment`'，分别来控制字符串相对于  $(x, y)$  的水平对齐和垂直对齐方式。例如，在 `mx+b` 模块的画图命令栏输入

```
>> text (0.5, 0.5, 'mx+b', 'horizontalAlignment', 'center')
```

请读者仔细体会这种设置属性值的方法，先指定要设置的属性名——'`horizontalAlignment`'，再紧接着设置它的值——'`center`'。这种方法在用命令设置模块属性时会经常使用，这样的好处是避免繁琐的输入参数对应，就以 `text` 命令而言，可以不设置 '`horizontalAlignment`' 属性，直接设置 '`verticalAlignment`'，也不会造成混乱。

Text 命令支持的水平对齐选项有（表 4-9）。

表 4-9 text 命令的水平对齐选项

选 项	对 齐 方 式
left	text 的左端处于指定点
center	text 的中点处于指定点
right	text 的右端处于指定点

表 4-10 列出了 `text` 命令支持的垂直对齐选项。

表 4-10 垂直对齐选项

选 项	对 齐 方 式
base	text 的基线处于制定点
bottom	text 的底线处于制定点
middle	text 的中线处于制定点
cap	text 的首线处于制定点
top	text 的顶线处于制定点

Printf 命令能够在图标的中心位置显示格式化了了的文本（`text` 参数或 `variablename` 的内容），文本的格式遵循参数“`format`”的设定。

要把文本分成几行显示，可以使用 '`\n`' 来表示断行（图 4-37）。例如

```
>> disp(' SampleMask')
```

```
>> disp(' Sample\nMask')
```

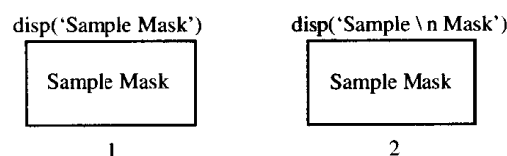


图 4-37 将文本分行

`port_label` 命令可以让用户指定显示在图标上的标签名，这个命令的语法为

```
port_label (port_type, port_number, label)
```

其中，`port_type` 的值可以取 ‘input’ 或者 ‘output’，`port_number` 是一个整数，`label` 说明这个端口的标签。例如，命令

```
port_label ('input', 1, 'a')
```

定义了端口 1 的标签为 ‘a’。

## 2. 在图标上显示图形

这里的显示图形，实际上指在图标绘图。完成绘图功能的函数有 `plot`，它的形式下面两种：

```
plot (Y);
```

```
plot (X1, Y1, X2, Y2, ...);
```

关于 `Y`, `X1`, `Y1`, ... 等等参数的意义和 MATLAB 命令函数中的差不多。只是要注意，在绘制图标时，`plot` 不支持使用不同的颜色、线型和标记点的选择。如果用户试图使用这些功能时，例如

```
>> plot ([0 1 5],[0 0 4], 'r')
```

这个表达式在 MATLAB 命令窗口输入，是没有任何问题的。但在画图命令栏输入，则显示的图标是三个问号的形式。它们表示所使用的绘制命令出现了问题，可能是以下几种情形：

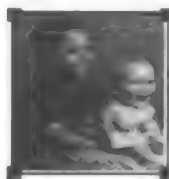
(1) 绘图命令用到的参数尚未定义（例如，当封装刚刚建立，还未在对话框输入参数的值）。

(2) 模块参数或绘图命令输入不正确。

## 3. 在图标上显示图像

用绘图命令绘制的图形毕竟很简单，而且绘制过程还很麻烦。最方便的途径是直接把图像显示在图标上，Simulink 向用户提供了这种能力。

在 MATLAB 里用来显示图像的命令是 `image`。它要求输入的参数是一个存储图像 RGB 三色值的数组，而不是图像的文件名。因此，使用 `image` 之前，必须先把图像文件转化为 MATLAB 的一个数组。这就需要用到命令 `imread` 或者 `ind2rgb`，读取位图文件并把它转化成 `image` 命令必须的格式。



在图标上显示图像

图 4-38 在图标上显示图像的效果

图 4-38 是把一幅图显示在模块的图标上的效果。

其产生办法是，首先在初始化命令栏里读入数据，并把它转化成 RGB 格式的数组。例如，要把 kids.tif 文件显示在图标上，可以这样写命令（图 4-39）：



图 4-39 在图标上显示图像的命令

其中，kids.tif 是 MATLAB 附带的一个图像文件可以在 toolbox\image\indemos 目录下找到，由于该目录处于 MATLAB 的搜索路径，所以不需要把绝对路径写出来。

然后再在绘图命令栏输入

```
>> image (x);
```

这里之所以要把前面的两个命令写在初始化命令栏，因为很多函数，如这里的 imread 在绘图命令栏里无法识别。除了在本小节里提及的用于画图标的命令，其他的命令在绘图命令栏都是无法识别的，即使是最简单的赋值运算。读者可以自己体会这一点。

制作图标时，image 支持的其他使用格式还有

```
image (a,[x,y,w,h])
```

```
image (a,[x,y,w,h],rotation)
```

#### 4. 在图标上显示传递函数

Simulink 里，有许多用于处理离散信号的模块，它们的图标往往用对应的传递函数来表示，就像图 4-40 所示的一样。

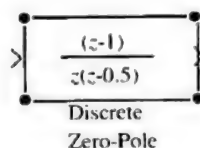


图 4-40 在图标上显示传递函数

这种图标显然不能用前面的方法得到。在图标上显示传递函数，可以使用下面的命令

```
dpoly (num,den)
```

```
dpoly (num,den,'character')
```

其中，num 和 den 分别表示传递函数分子多项式和分母多项式的系数，属于向量类型；而“character”用来指定多项式表达式中的字母形式，通常缺省值是“s”，表示对应的模块是个连续系统。具体地讲，可以分为以下几种情形：

- (1) 按字母 s 的次数的降序排列，显示连续的传递函数。使用 dpoly (num,den)。

例如,  $\text{num}=[0\ 0\ 1]$ ,  $\text{den}=[1\ 2\ 1]$ , 显示的图标为

$$\frac{1}{s^2 + 2s + 1}$$

(2) 按字母  $z$  的次数的降序排列, 显示离散的传递函数。使用 `dpoly (num,den,' z')`。还是用前面的 `num` 和 `den`, 则显示为

$$\frac{1}{z^2 + 2z + 1}$$

(3) 把以上的按升序排列, 只需把  $s$  和  $z$  变成  $s$ -和  $z$ -即可。

(4) 按函数的零极点来显示。使用命令 `droots`。

例如, `droots ([1], [1,2], 1)`

#### 5. 控制图标属性

用户还可以设置一些参数来控制图标的属性, 这些参数在 `icon` 页的右下端, 设置的方式是从 `popupcontrol` 中选取一个选项。这些参数的作用解说如下。

(1) `icon frame` 指包含模块的正方形框, 可以通过设置这个参数为 `visible` 或 `invisible`, 分别使得正方形外框可见或隐藏。

(2) `Icon transparency` 参数用来设置图标的透明性, 它有两个选项: `Opaque` 和 `transparent`。`Opaque` 表示图标是不透明的, 这时它将会覆盖以前 Simulink 自动在模块上显示的文字和端口标签。反之, 如果设成 `transparent`, 则图标是透明的, 不会覆盖端口标签等。缺省设置是 `Opaque`。

(3) `Icon Rotate` 参数用来控制图标的朝向是否为固定, 也有两个选项: `fixed` 和 `rotate`。当选择为 `fixed` 时, 当模块被旋转或翻转时, 图标的方向保持不变; 而设置为 `rotate` 时, 图标将随着模块一起变化。也就是说, 这两个属性实际上分别对应着绝对方向和相对方向保持不变。

(4) `Drawing coordinates` 参数的作用是调整图标绘制坐标。它的三个选项值意义分别是: `pixel` 表示按模块的实际大小调整模块方框的大小, `autoscale` 表示按所画图标的实际大小来调整模块方框的大小, 而 `normalized` 则是把绘制图标的坐标系归一化。

#### 4.5.4 documentation 页

封装编辑框的 `documentation` 页, 是最简单的一页了, 它各个部分的意义十分明确。这里只着重强调两个地方:

首先, `Mask type` 这一项仅仅是为了文档说明的目的, 没有特殊的意义。这和在建模时, 模块下的标签不同, 它是作为模块的名称存储在模型文件里的, 是用来标识模块的, 所以在模型里必须是互异的。这里的 `Mask type` 可以是为该封装选择的任何名称, 它将会显示在封装后的模块对话框里。

其次, 还请读者注意 `Mask help text` 这一栏。它主要是用来定义模块对话框的 `help` 按钮

被按下后出现的帮助文档。它支持的格式有：

- (1) URL 说明（以 `http:`、`www:`、`file:`、`ftp:`等开始的字符串）；
- (2) Web 命令；
- (3) eval 命令；
- (4) 显示在 Web 浏览器的静态文本。

要显示帮助时，Simulink 首先检查 Mask help text 的第一行，如果检测到 URL 说明、Web 命令或者 eval 命令，Simulink 将直接访问这些命令定义的帮助文件；否则把 Mask help text 里的所有内容显示在浏览器里。

下面是几种能被 Simulink 接收的命令。

```
web ([docroot '/My Blockset Doc/' get_param('gcb','MaskType') '.html'])
eval ('!Word My_Spec.doc')
http://www.mathworks.com
file:///c:/mydir/helpdoc.html
www.mathworks.com
```

#### 4.5.5 为封装的模块建立动态对话框

Simulink 支持用户建立随着用户输入参数变化的模块参数对话框。封装对话框的特性可以改变的方式，包括：

- (1) 参数控件的外观。

改变一个参数可以导致另外一个参数对应的控制出现或者消失。而在一个空间出现或者消失时，对话框也会相应的扩张或者缩小。

- (2) 参数控件的使能状态。

改变一个参数可以导致另一个参数的控件使能或者禁止输入。当一个控件被禁止时，它在外观上就变灰。

- (3) 参数值。

改变一个参数的值可以致使相关联的参数被置为合适的值。

建立一个封装对话框必须将封装编辑器和 Simulink `set_param` 命令结合起来使用。具体地说来，首先，要在封装编辑框定义所有的对话框参数，包括静止的和动态的。接着，在 MATLAB 命令行使用 Simulink 的 `set_param` 命令，来指定定义对话框对输入响应的回调函数（callback functions）。最后就可以保存包含被封装的子系统的模型或者库来完成动态封装对话框的建立。

##### 1. 设置封装模块对话框参数

Simulink 定义了一系列的封装模块参数来定义封装对话框的当前状态。用户可以使用封装编辑器来查看或者设置这些参数中的多数。同样可以使用 `get_param` 和 `set_param` 命令来查看和设置对话框参数。使用 `set_param` 命令的好处在于，它可以在对话框被打开时设置参数，并立即改变对话框的外观。这也就让用户建立动态封装对话框。

例如，读者可以在 MATLAB 命令行使用 `set_param` 命令来设定当用户自定义的模块参数被改变时，Simulink 要调用哪个回调函数来处理这种变化。而被调用的回调函数就依次地使用 `set_param` 命令来改变封装对话框自定义参数的值或者它的状态，如隐藏（hide），显示



(show), 使能(enable)或者禁止(disable)用户自定义参数控件(关于回调函数请见第五章)。

## 2. 预定义封装对话框参数

Simulink 中与封装对话框有关的预定义参数有:

### (1) MaskCallbacks

这个参数的值是一个字符串的单元数组, 用来指定对话框用户定义参数控件各自的回调表达式。其顺序按照参数定义的顺序和单元一一对应, 即第一个单元定义了第一个参数控件的回调函数, 第二个单元对应第二个参数控件等等。回调可以是任何有效的 MATLAB 表达式, 包括调用 M 文件的表达式, 这就使用户实现复杂的回调。

为封装对话框设置回调最容易的方法是, 首先在模型或者库窗口选择相应的封装模块, 然后在 MATLAB 命令行键入 set\_param 命令。如下面的代码:

```
>> set_param(gcb,'MaskCallbacks',{'parm1_callback','','parm3_callback'});
```

这个命令定义了选中模块的封装对话框的第一个和第三个参数的回调函数。最后别忘了保存包含封装模块的模型或者是库来保存回调设置。

### (2) MaskDescription

这个参数的值是一个用于设定模块描述的字符串, 它的作用就是设置它, 可以动态的改变模块描述。

### (3) MaskEnables

这个参数的值是一个字符串的单元数组, 它定义了用户定义参数控件的使能状态。它的对应顺序和 MaskCallbacks 参数相同, 即第一个单元对应第一个参数, 以此类推。on 代表这个控件可以被用户输入; off 表示控件被禁止。于是用户可以在回调中动态的使能或者禁止用户的输入。例如, 下面的命令

```
>> set_param(gcb,'MaskEnables',{'on','on','off'});
```

这个命令在封装对话框一被打开时, 就会禁止第三个控件。

### (4) MaskPrompts

这个参数的值是一个字符串的单元数组, 用来指定用户定义参数的提示, 第一个单元对应一个参数, 以此类推。

### (5) MaskType

这个参数的值是与对话框相关的模块的封装类型。

### (6) MaskValues

这个参数的值是一个字符串的单元数组, 用以指定用户定义参数的值, 对应顺序同上。

### (7) MaskVisibilities

这个参数的值也是一个字符串单元数组, 用以指定用户定义参数控件的可视与否, 对应顺序同上。on 表示相应的控件是可视的, off 表示控件是隐藏的。例如,

```
>> set_param(gcb,'MaskVisibilities',{'on','off','on'});
```

## 4.6 建立条件子系统

条件子系统是指它的执行受输入信号控制的那一类子系统，在条件子系统里，控制子系统是否执行的信号被称为控制信号。进入子系统里的信号被称为控制输入。

条件子系统在建立复杂的模型时非常有用，因为这些模型的某些组件要受模型内其他的组件控制的。例如，数字电路里的计数器，就是在每个时钟到达时刻将计数器的值增加一个。像这样的受控子系统，在数字电路里还可以找到很多。所以说，建立条件子系统在实际建模中经常会遇到。

Simulink 支持三种类型的条件子系统：

(1) 使能子系统。当控制信号为正时，它才执行。执行时间从控制信号从负变到正（过零点）的时间步开始，并在控制信号为正的时间内持续执行。它也就是数字电路里所谓的电平触发。

(2) 触发子系统。在发生触发事件的时刻执行。触发事件在 Simulink 里是用触发信号的上边沿和下边沿来表示的。

(3) 触发使能子系统。当控制信号为正时，如果触发事件发生，则子系统执行。

### 4.6.1 使能子系统

使能子系统在控制信号为正值的时间步里执行。每一个子系统只有一个控制输入，它既可以是标量信号，也可以向量信号。当为标量时，只要该信号大于零，子系统就开始执行。若控制输入为（注意控制输入和控制信号的区别）向量时，只要其中一个信号大于零，那么子系统也执行。

Simulink 在判断是否执行时，主要采用的是过零检测技术（将在第五章提到）。只要检测到了过零点，并且斜率为正，则子系统开始执行。如果检测到信号滑过零线，且斜率为负的，则子系统停止执行。

生成一个使能子系统的方法很简单。只要把一个 enable 模块复制到子系统模块中即可，它的位置是在 Simulink 库的 signals&systems 子库里。图 4-41 是添加了 enable 模块后的子系统的外形。

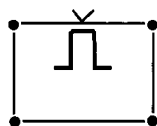


图 4-41 使能子系统图标

其余的操作和生成一般的子系统区别不大，或者说在这里 enable 更像是一个标记符号，它告诉 Simulink，该子系统将受到从这个端口输入的信号的控制。至于怎么控制，由 Simulink 来处理，用户不需考虑如何设计实现这一点。所以 enable 模块在子系统的图表上是一个独立的，与别的模块没有任何的连线。用 Simulink 的术语讲，enable 模块就是虚模块。

以一个半波整流系统为例。它的作用是在输入信号为正时，输出原信号，而在信号为负

时输出为零。设计这个系统，就没必要去考虑如何判断输入信号的正和负，使用 enable 模块很容易就能做到这一点，只是这里输入信号和控制信号是同一信号罢了。模型图表如图 4-42 所示。

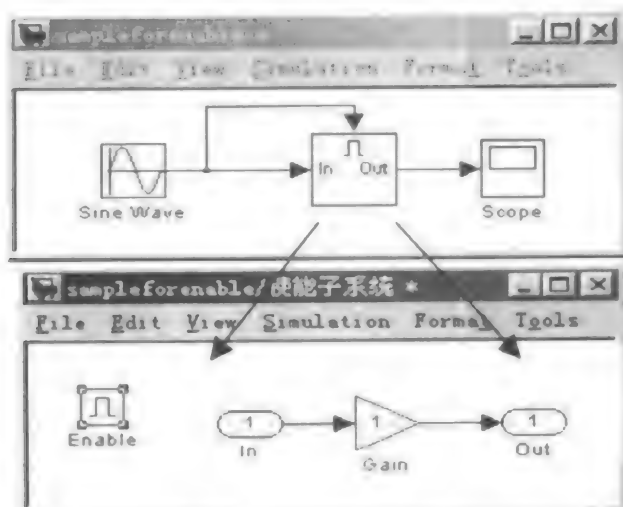


图 4-42 半波整流模型框图

经过上面这个例子的切身体验，读者对 Simulink 里建立使能子系统的步骤应该有了初步的了解。但是，使能子系统和普通的子系统还是有些不同。

由于使能子系统在控制信号为负值时，是不执行的，所以设计子系统时要设置当子系统不执行时的系统输出。为此，设计时要设置系统输出端口的参数。请双击打开输出端口的参数对话框（见图 4-43）。

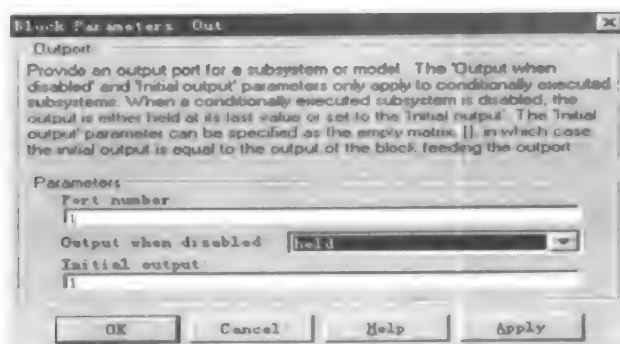


图 4-43 输出端口参数对话框

当 Output when disabled 参数选择为 held，则该端口的输出保持最近时刻的输出值；如果为 reset，则输出被置为它的初始输出。而 initial output 参数定义了这个初始值。

在设计条件子系统中，另外一个要考虑的问题是，在系统被禁止时，是让系统保持上次执行的状态值，还是重新置为系统的初始状态。这里，关于状态的概念将在第五章讲解。

为此，用户需要设置 Enable 模块的参数。图 4-44 显示了 Enable 模块的参数对话框。图中，选项 held 和 reset 的意义和前面是类似的。上面 show output port 检查框的作用是允许系统输出控制信号，因为在某些场合下，控制信号还有别的用途。

除了上面的一些要求外，使能子系统可以包含任何模块。

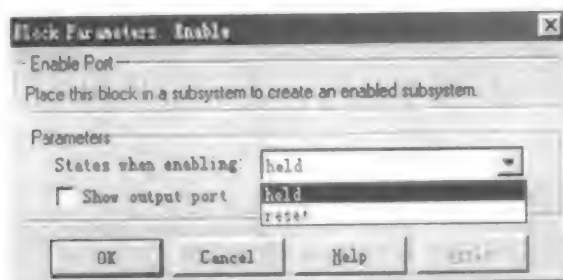


图 4-44 enable 模块的参数对话框

### 4.6.2 触发子系统

触发子系统只在触发事件发生的时刻执行。触发事件是由控制输入信号的状态来决定的，一个触发子系统只能有一个控制输入，它在 Simulink 里被称为触发输入。

触发事件有两种类型：上升触发和下降触发。所谓的上升触发事件，指控制信号从负值或零变为正值，也可以是从负值变为零。而下降触发事件，则是指控制信号从正值变为零或负值，或者是从零变成负值。Simulink 允许用户选择子系统是在上升触发事件发生时执行，还是在下降触发事件发生时执行，或者是两者之一发生时都执行。

例如，下面的控制信号，它的触发事件发生示意图为图 4-45，其中 R 表示上升触发事件，F 表示下降触发事件。

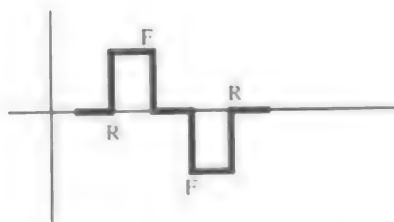


图 4-45 触发事件示意图

至于 Simulink 是如何检测触发事件是否发生，本书的第五章将会讲到。

和建立使能子系统一样，在 Simulink 里，建立触发子系统的过程也是十分简单的。类似的，只要向子系统内加入一个 trigger 模块就可以了，它的位置也是在 Simulink 库下的 signal&system 子库里面。图 4-46 是一个简单触发子系统示例，而右边的小图则是触发子系统的内部结构。

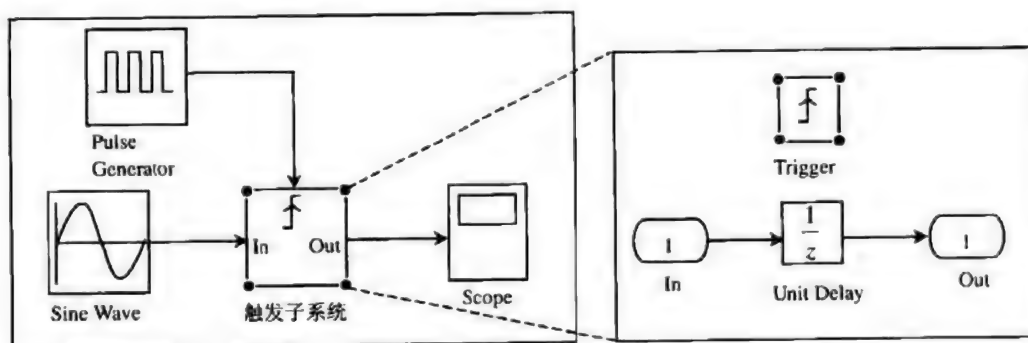


图 4-46 简单的触发子系统示例

读者可以仿照上图建立这个模型，其中 pulse generator 在 source 子库里可以找到，它在这里产生一个时钟基准信号。在运行模型之前，请把 unit delay 模块参数对话框的 sample times 改成-1，它表明该模块的采样时间是继承的，受驱动模块的控制。关于继承的具体意义，请见本书的第五章或者第九章。

不难看出在这个模型里，触发子系统起到了一个离散保持采样器的作用。为了把结果看得更清楚，可以用一个 mux 模块把正弦波、脉冲信号和模块信号合成在 scope 上显示。显示的结果如图 4-47 所示。

同使能子系统不同的是，触发子系统在两次触发时间之间，通常是保持最近的输出和状态的，即处于保持状态。读者在设置输出端口的参数时，会发现 output when disabled 这一参数无法设置，而 trigger 模块的参数对话框里根本就没有类似与 enable 模块的设置状态保持与否的项。在 trigger 模块里，允许用户设置模块的触发类型：rising、falling、either 和 function call。前面三个的意义是十分明显的，分别指在子系统在上升触发事件、下降触发事件和两者之一发生都执行。前面的例子，就是上升触发类型的。读者可以自己尝试一下不同的事件类型对仿真结果的影响。至于 function call，这里只略微提一下，它与 S-函数的编写有关。简单地说，函数调用子系统，是指该触发子系统的执行不是由控制信号来决定，而是由一个 S 函数的内部逻辑来决定。什么是 S 函数，请看 S 函数一章。



图 4-47 触发子系统运行结果

触发子系统只能在仿真的特殊时刻才执行，因此，对能包含在触发子系统里的模块有一些要求：

- (1) 那些具有继承的采样时间的模块，如 gain 模块或逻辑运算模块；
- (2) 采样时间被设为-1 的离散模块，这表示它的抽样时间从驱动模块继承过来。

#### 4.6.3 触发使能子系统

第(3)种条件子系统是前面两种条件子系统的结合，在 Simulink 里，被称为触发使能子系统。可以从名称的字面意思来理解这种子系统的行为，在触发使能子系统里，使能的不是系统的执行，而是对触发功能的使能。图 4-48 是它的流程图表示。

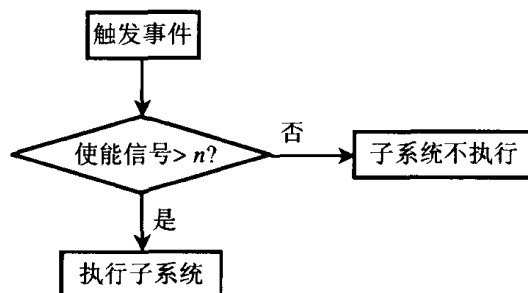


图 4-48 触发子系统流程图

在触发使能子系统里，触发控制信号和使能触发信号是分开的。当触发事件发生时，Simulink 就检查使能端的控制输入信号，如果它大于零，Simulink 就执行子系统。当两个输入信号都是向量信号时，只要向量信号有一个元素非零时，子系统就执行。

建立一个使能子系统，只要把 **enable** 模块和 **trigger** 模块都添加到子系统里就可以了。因为这里有 **enable** 模块的存在，所以在建立触发使能子系统时，也要像建立使能子系统一样，设置当子系统在禁止时的输出和状态。实际上，在触发使能子系统里，**enable** 模块和 **trigger** 模块相互独立，所以它们的设置互不干扰。因此，可以分别按前面讲的方法对它们进行设置。

前面举过一个半波整流的例子，下面将来看看实现全波整流。最基本的思路是，让两个子系统在输入信号大于零和小于零的两种情况下分别执行，然后把它们的结果合并起来。对于大于零的情况直接使用前面的半波整流子系统，而对于信号小于零的情况，只需把输入信号乘一个 -1，就可以利用前面的半波整流子系统了。在 Simulink 里，完成合并功能的模块是 **merge** 模块，请注意这个模块和 **mux** 的区别。图 4-49 是最后建好的模型。

其中，**merge** 的模块是把两个信号合成一个信号，它与 **mux** 不同，**mux** 是一个虚拟模块，只是把多个信号合并成一个 **size** 更大的向量信号，或者说只是空间上的合并。而 **merge** 模块则是把多个信号在时间上合成一个信号，新信号在不同时间上的取值来自不同的源信号，也就是常说的时分复用，这是一个非虚拟模块。读者可以比较它们的输出，来体会这种区别。而图中的正向半波整流和反向半波整流都是和前面提到的半波整流同样的内部结构。

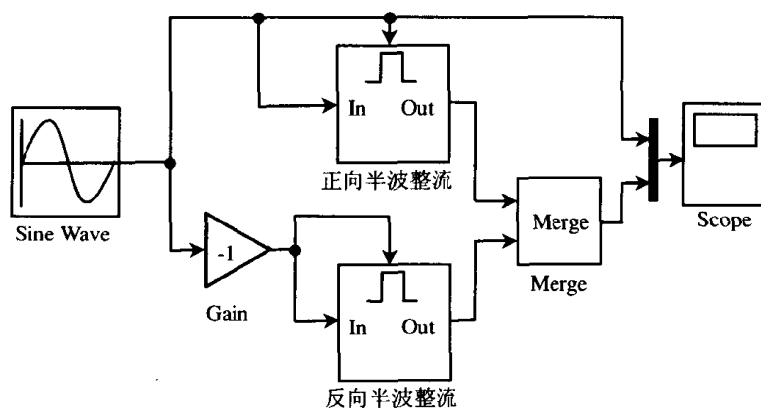


图 4-49 全波整流系统

运行模型后，输出的结果为图 4-50。

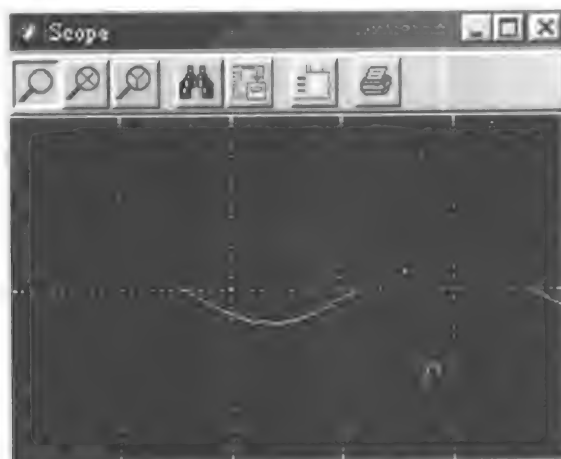


图 4-50 输出的结果

## 第五章 深入理解 Simulink

### 5.1 Simulink 如何工作

经过前面的学习，大多数的仿真问题，读者都可以应付了。但是要想灵活高效的使用 Simulink，就必须对它的工作原理有些了解。尽管 Simulink 的初衷是为用户屏蔽掉许多繁琐的编程工作，而把主要精力放在模型的构建上。

#### 5.1.1 基本模型

简单地说，Simulink 里的每一个模块都是第一章中所说的一个系统，它有输入、输出和状态三个基本元素。在 Simulink 里，模块都是用向量来表示这三个元素的，本书分别用  $u$ 、 $x$  和  $y$  来标记输入、状态和输出向量。图 5-1 反映了这个关系。

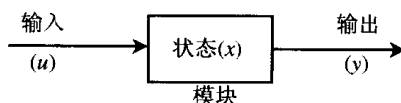


图 5-1 Simulink 模块的基本模型

在图中的三个基本元素中，状态向量无疑是最重要的，也是最灵活的一个概念。何谓状态呢？读者首先想到的可能是在线性系统理论里用来表示连续方程用的状态空间方法里的状态，在那里，状态被定义成一些微分。可以说那里的状态只是状态这个概念内涵的一部分，不仅仅只是工程系统存在状态，事实上，状态在非工程系统也能找到它的对应物。例如，前面讲到的排队系统中，队列中的等候人数等都可以被理解为状态。

确切地说，状态决定了模块的输出，而它的当前值是前一个时间的模块状态和（或）输入的函数。拥有状态的模块必须保存前面的状态值，并计算出当前的状态值。具有状态的模块也就拥有保存以前状态值或者输入值的存储空间。Simulink 的 Integrator 模块是有状态模块的一个例子，Integrator 模块输出的是输入信号从仿真开始时到当前时刻的积分值。当前积分值依赖于 Integrator 模块的输入的历史纪录，因此积分值是模块的一个状态。而 Gain 模块则是无状态模块的例子。Gain 模块的输出完全由当前的输入值和增益决定，因此，Gain 模块没有状态。

在 Simulink 里状态向量可以分为连续状态、离散状态以及两者的结合。输入、输出和状态这三个量的关系可以用下面的方程来反映。

$$y = f_o(t, u, x)$$



$$x_{d_{k+1}} = f_u(t, u, x)$$

$$x'_c = f_d(t, u, x)$$

$$\text{其中, } x = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$$

上面三个式子的意义是十分直观的。无论是对于连续系统还是对于离散系统，在用计算机进行仿真时，要估算一个系统的输入、输出以及状态向量，都是在采样时间点进行的，也就是仿真时间步。在每一个采样时刻，**Simulink** 根据当前的时间、输入和状态来决定该采样时刻的输出，这个关系用  $f_o$  来表示。对于离散状态，则要根据以前的状态来计算当前状态，对于一个行为复杂的系统，离散状态的更新完全有可能和连续状态有关系，所以式中是指整个系统的状态。而对于连续状态则是计算出连续状态当前的微分值。也就是说任何一个模块的行为，都可由上式的三个函数来刻画，设计一个模块，也就是要确定这三个函数，当然首先要确定三个向量输入、输出和状态的意义。

### 5.1.2 进行仿真

**Simulink** 仿真包括两个阶段：初始化和模型执行。

#### 1. 初始化

在初始化阶段，要完成的工作有：

- (1) 模块参数传给 **MATLAB** 进行估值，得到的数值结果将作为模块的实际参数。
- (2) 模型的各个层次被展开。每一个不是条件执行的子系统被它所包含的模块替代。
- (3) 模型中的模块按更新的次序进行排序。排序算法产生一个列表确保具有代数环的模块在产生它的驱动输入的模块被更新后再更新。当然，这一步也就要先检测出模型中存在的代数环。

(4) 决定模型中没有显式设定的信号属性，例如名称、数据类型、数值类型以及大小等，并且检查每个模块是否能够接受连接到它们输入端的信号。

**Simulink** 使用一个名为属性传递的过程来决定未被设定的属性，这个过程将源信号的属性传递到它所驱动模块的输入信号。

- (5) 决定模型中所有没有显式设定采样时间的模块的采样时间。
- (6) 分配和初始化用于存储每个模块的状态和输出的当前值的存储空间。

#### 2. 模型执行

完成这些工作之后就可以运行仿真了。一个模型是使用数值积分来仿真的。所运用的仿真解法器（仿真思想）依赖于模型提供它的连续状态的微分能力。计算微分可以分成两步来进行：首先，按照排序所确定的次序计算每个模块的输出；然后再根据当前时刻它的输入、状态来决定状态的微分，得到微分向量后再把它返回给解法器，后者用它来计算下一个采样点的状态向量。一旦新的状态向量计算完毕，被采样的数据源模块（sine wave 模块等）和接收模块（scope 模块等）才被更新。

在仿真开始时，模型设定待仿真系统的初始状态和输出，在每一个时间步，**Simulink** 计算系统的输入、状态和输出，并更新模型来反映计算出的值。在仿真结束时，模型得出系统

的输入、状态和输出。

在每个时间步，Simulink 所采取的动作依次为：

(1) 按排列好的次序，更新模型中模块的输出。Simulink 通过调用模块的输出函数计算模块的输出（见第九章）。Simulink 把当前值，模块的输入和状态传给这些函数计算模块的输出。对于离散系统，Simulink 只有在当前时间是模块采样时间的整数倍时，才会更新模块的输出。

(2) 按排列好的次序，更新模型中模块的状态。Simulink 计算一个模块的离散状态的方法是，调用模块的离散状态更新函数。而对于连续状态，则是对连续状态的微分（在模块可调用的函数里，有一个用于计算连续状态的微分）进行数值积分来获得当前的连续状态。

(3) 额外的检查模块连续状态的不连续点。Simulink 使用一种称为过零点检测（zero crossing detection）来检测连续状态的不连续点。

(4) 计算下一个仿真时间步的时间。这是通过调用模块获得下一个采样时间函数来完成的。

### 3. 决定模块更新次序

在仿真中，Simulink 更新状态和输出，都要根据事先确定的模块更新次序，而更新次序对仿真结果的有效性非常关键。特别的，当模块的输出是它当前时刻的输入值的函数，那么这个模块必须在驱动它的模块被更新之后才能被更新，否则，模块的输出将是没有意义的。

这里请读者不要把模块保存到模型文件的次序和仿真过程模块被更新的持续相混淆。Simulink 在模型初始化将模块排好正确的次序。

为了建立有效的更新次序，Simulink 根据输出和输入的关系，将模块分类。其中，当前输出依赖于当前时刻输入的模块称为直接馈入模块，所有其他的模块都称为非虚拟模块（关于直接馈入，见 5.1.4）。直接馈入模块的例子有 Gain, Product 和 Sum 模块，非直接馈入模块的例子有 Integrator 模块（它的输出只依赖于它的状态），Constant 模块（没有输入），Memory 模块（它的输出只依赖于前一个时间步的输入）。

基于上述的分类，Simulink 使用下面的两个基本规则对模块进行排序：

(1) 每个模块必须在它驱动的任何模块中的任何一个之前被更新。这条规则确保模块在被更新时，它的输入有效。

(2) 非直接馈入模块可以按任何的次序更新，只要它们在它们所更新的直接馈入模块之前更新。这条规则可以通过把所有的非直接馈入模块以任何次序放在更新列表来满足。因此，它允许 Simulink 在排序过程中忽略非虚拟模块。

在排序过程中，Simulink 检查和标记代数环的出现。所谓代数环是指直接馈入模块的输出直接或者通过别的直接馈入模块连到模块的一个输入的信号环路。这样的一个环路在更新模块时会产生一个死锁。然而代数环可以用于表达一系列的代数方程，这里把模块的输入和输出作为未知数。并且，这些方程在每个时间都要有解。因此，Simulink 假定包含直接馈入模块代表一些有解的代数方程，并在每次模块被更新时，解出这些方程。

另外一个约束模块更新次序的因素是用户给模块设定优先级，Simulink 在低优先级模块之前更新高优先级的模块。

### 5.1.3 过零检测

Simulink 用过零检测来检测连续信号的不连续的地方。过零检测在以下几个方面扮演着重要的角色：

- (1) 状态事件的获取；
- (2) 不连续信号的精确积分。

#### 1. 状态事件的获取

一个系统发生一个状态事件是指，系统的某个状态值发生了能使系统产生显著变化的变化。状态事件的一个简单例子就是和地板相撞来回反弹的球。要对这样一个系统进行仿真，解法器不可能精确的使仿真步与球和地板接触的时间重合。因此，球就像是穿过了接触点，或者说球穿透了地板。

Simulink 使用过零点检测使仿真步精确地（在机器精度范围内）发生在状态事件发生的时刻。于是因为仿真的时间步准确的取在接触的时刻，所以仿真就不会产生穿透现象，并且速度从负到正的转换非常迅速。Simulink 里有一个弹球的演示模型，图 5-2 是它的图表，读者可以在 MATLAB 命令窗口输入 bounce 来演示它，或者也可以在 MATLAB 的 demo 窗口直接寻找。

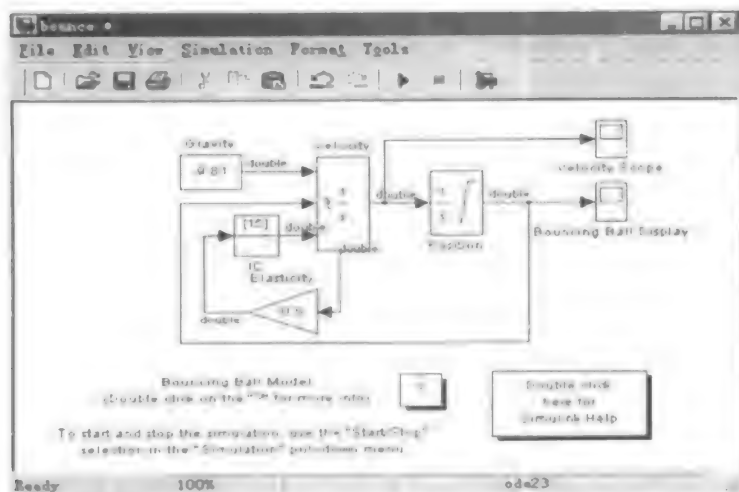


图 5-2 bounce 模型图表

下面对这个演示模型进行简单的说明。它模拟以一定的初速度向上弹的小球，反复与地板相撞并最终变成零的过程。仿真模型有两个基本假设：

第一个假设，不考虑空气阻力。在这个假设下，球在空中运行过程中的机械能量是不变的，也就是说球在与地板撞击后的瞬时速率和下次与地板撞击前的速率是相同的。

第二个假设，就是对球和壁板的撞击过程进行了简化。在这里，模型假定撞击后的速度与撞击前的速度之比始终是一个常数，在上图中是-0.8。

明白了这两个假设，下面来看看用 Simulink 建立该模型的具体技巧。根据牛顿运动定理，不难理解速度和加速度之间的积分关系以及位移和速度间的积分关系。所以在模型的实现里有两个积分器——名称分别是 Position 和 Velocity。对于 Position 模块，它的输入正是速度信号，并且由于球的初始位置是 0，所以它的初始状态是零。此外，球与地板接触与否对位移

的运算规律没有任何影响，所以 Position 积分器的积分限是 0 到无穷大。但对于 Velocity 积分模块，由于小球和地板接触之后速度改向，并且使积分的初始值变化，所以结构和常用的积分模块不太一样。从图中可以看出，这个模块有三个输入和两个输出。

请读者双击这些模块来看看它的参数对话框，就不难发现这些多余的输入和输出端口是如何添加的。图 5-3 是它的样子。

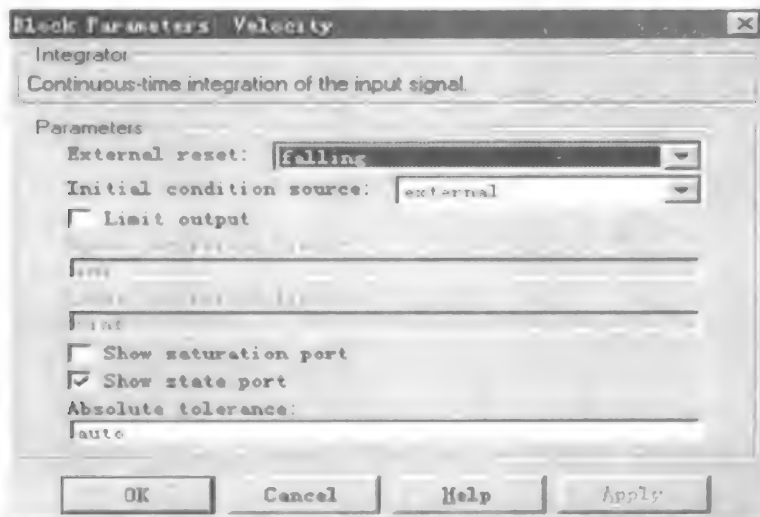


图 5-3 参数对话框

external reset 参数，表示由外部信号使积分重新回到初始值，选择它的效果就是 Position 模块中类似于下降触发器的标志。而 initial condition source 参数在这里被设为了 external，这表示积分的初始值可以由外部输入，一个好处就是可以动态的改变积分的初始值，选择它的效果就是 Position 模块左下角增加了一个输入端口。这个端口由一个 initial condition 模块来驱动。而检查框 show state port 被选中的直接效果就是右下角的输出端，它输出积分的状态，对积分器而言就是积分的输出，这个输出经过一个增益为-0.8 的 Gain 模块输入到 Initial Condition 模块。实际上 Gain 模块就模拟了小球与地板接触后对速度的影响。

这个模型的最大的困难就是如何确定小球与地板接触，在 Simulink 里是通过过零检测来解决它的。读者可以看到，Position 积分器的输出输入到 Velocity 的外部 reset 端口，因为该端口是下降触发的，一旦位移从正变为负就产生了一个下降触发事件，Simulink 可以通过过零检测来捕获这个事件。于是一旦检测到改下降触发，就会使 Velocity 重新置为初始值，而此时的初始值为当前状态值乘以-0.8，这就和物理过程符合了。

读者可以运行仿真，来看看仿真的结果。

## 2. 不连续信号的积分

数值积分方法是建立在所积分的信号是连续的，并且具有连续的微分的假设下。如果在一个积分过程中遇到了不连续点（状态事件），Simulink 使用过零检测来寻找不连续点发生的时间。而这次积分的上限只取到不连续点的左边沿。最后，Simulink 略过不连续点，并对信号的下一段分段连续进行积分。

Simulink 模块中使用过零检测的一个例子是 Saturation 模块。过零点检测 Saturation 模块中的这些事件：

- (1) 输入信号到达上限;
- (2) 输入信号离开上限;
- (3) 输入信号到达下限;
- (4) 输入信号离开下限。

定义了自己的状态事件的 Simulink 模块被认为是具有固有的过零点。如果用户需要一个过零点事件的显示通知, 可以使用 hit crossing 模块。

状态事件的检测依赖于一个内部过零点信号的构建。这个信号是不能被模块图表理解的。就 Saturation 而言, 它用于检测上限的过零点信号是  $zcSignal = UpperLimit - u$ , 其中  $u$  是输入信号。

过零点信号有一个方向属性, 它的取值有三种:

- (1) rising—— 当一个信号上升到零或者穿过零, 或者离开零并且变成正数时发生的过零点;
- (2) falling—— 当一个信号下降到零或者穿过零, 或者离开零并且变成负数时发生的过零点;
- (3) either—— rising 或 falling 有其一发生时就发生。

对于 Saturation 模块的上限而言, 过零点的方向是 either。这样使得信号进入饱和或者离开饱和的事件可以通过相同的过零信号来检测到。

误差限度的大小对过零点的检测有很大的影响。如果误差限度太大, Simulink 就有可能检测不到过零点。数学里有这样一条定律, 对于连续信号, 如果有两个点的符号相异, 那么在这两点之间必定存在着一个过零点。这个定律是 Simulink 检测过零点的数学基础, 它通过检查一个仿真时间步的首尾两点的符号, 来判定在该时间步那里是否存在过零点。但如果误差限度太大, 就导致仿真的时间步不是足够的小, 有些过零点就不能检测出来了。图 5-4 显示了仿真时间步的大小对过零点检测的影响。

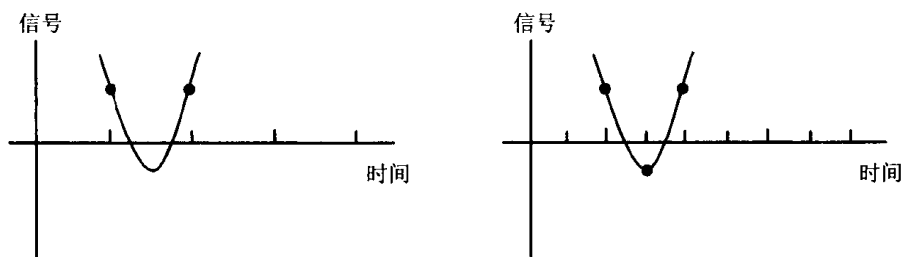


图 5-4 过零点示意图

上图中, 左图的仿真时间步取得较大, 所以对于图中的信号在仿真步的首尾的采样值都是正数, 没有符号变化, 于是 Simulink 就会漏掉这两个过零点。而右图的仿真时间步缩小为原来的一半, 就顺利的检测出这个过零点来。

为了防止这种情况的出现, Simulink 允许用户修改误差限度(见第六章的仿真参数设置), 通过缩小误差限度, 可以使 Simulink 采用更小的仿真步长。

## 5.2 代数环

### 5.2.1 直接馈入环路 (direct feedthrough)——代数环

在 Simulink 里直接馈入的概念是指, 具有直接馈入的模块在不知道输入端口的值的前提下无法计算输出端口的值。也就是当前时刻输出值的计算依赖于当前时刻的输入值, 从模块的内部结构上说, 模块内不存在延迟单元。在 Simulink 里, 直接馈入的模块有:

- (1) Elementary Math 模块;
- (2) Gain block 模块;
- (3) Integrator 模块的初始条件端口;
- (4) Product 模块;
- (5) 有非零的 D 矩阵的 State-Space 模块;
- (6) Sum 模块;
- (7) 分子和分母多项式具有相同阶数的 Transfer Fcn 模块;
- (8) 零点数和极点数相同的 Zero-Pole 模块。

前面几个模块为什么是具有直接馈入的模块, 是很容易理解的。这里只略微提一下 (4), (6), (7) 三个模块。读者可以从库里找到 State-Space 模块, 查看它的参数模块的描述信息。State-Space 模块的输出  $y=Cx+Du$ , 在 D 不为零时, 输出 y 就和相同时刻的输入信号有关了, 自然是直接馈入了。也就是说, 直接馈入中的直接是指仿真时间意义上的, 而不是指是否经过处理, 即和直接连线不是等同的。而对于 Transfer Fcn 模块和 Zero-Pole 模块附带的要求, 都是为了保证模块所对应的传递函数具有常数项, 这样当前时刻的输出就和当前时刻的输入有关了, 所以也具有直接馈入。

对于大多数的模块, 读者都可以按照上面的原则自行判断, 如果实在是无法确定, 可以查阅本书第八章的模块参考。

当具有直接馈入的端口由该模块的输出驱动时, 或者是经过别的具有直接馈入的模块的反馈环路驱动, 就发生了代数环。例如, 图 5-5 所示的就是一个代数环的例子。

在图中, sum 模块是具有直接馈入的模块, 它的输出直接连到 sum 模块的输入端。于是整个模型所表示的意思就是一个方程了:

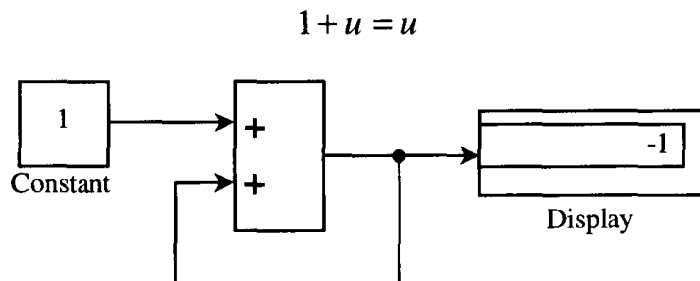


图 5-5 代数环示例

当然, 这个方程是无解的, 所以当读者试图运行这个模型时, Simulink 就会显示错误信

息。为此，可以在中间添加一个增益模块，就像图 5-6 所示的一样。

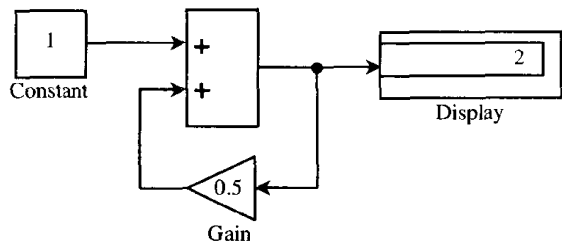


图 5-6 改进后的模型

在 Simulink，很容易建立具有多代数状态变量的向量代数环，下面的模型就是一个具有变量  $z_1$  和  $z_2$  的代数环。

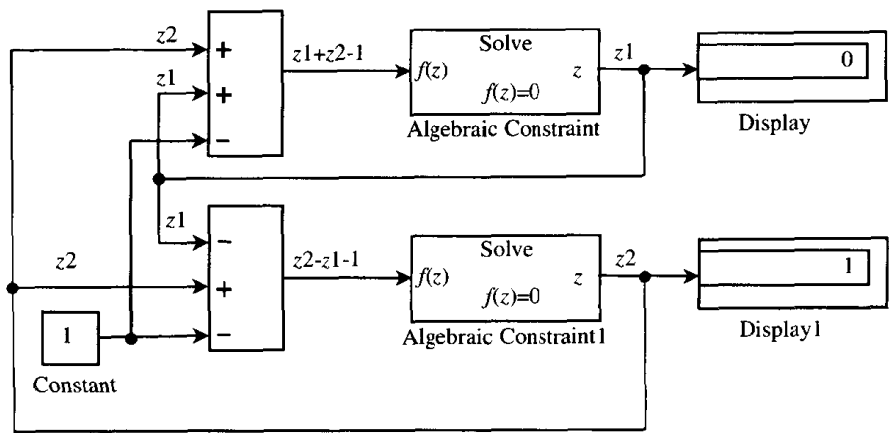


图 5-7 多状态代数环

图 5-7 的示例模型的作用就是通过建立一个向量代数环来解一个二元一次方程组：

$$\begin{cases} z_1 + z_2 - 1 = 0 \\ z_2 - z_1 - 1 = 0 \end{cases}$$

读者要注意模型中，algebraic constraint 模块的用法，它的位置是在 math 子库。algebraic constraint 是对代数方程建模的便捷方法，algebraic constraint 模块将它的输入信号  $F(z)$  约束为零，并输出一个代数状态  $z$ 。注意 algebraic constraint 使用时，输出信号必须通过某种反馈途径影响输入信号，为了提高代数环解法器的效率，algebraic constraint 模块允许用户在参数对话框里设置它的初始猜测值。

当一个模型包含代数环时，Simulink 就在每一个时间步调用一个环路解法器的方法。环路解法器用迭代的方法来决定代数环所对应的方程。因此，具有代数环的模型比没有代数环的方程运行得慢。

为了解决  $F(z)=0$ ，Simulink 解法器使用具有弱的线性搜索的秩为 1 的牛顿方法更新偏微分 Jacobian 矩阵。尽管这个方法是鲁棒的，但如果代数状态  $z$  没有一个好的初始值估计，解法器就有可能不会产生一个收敛的值。除了可以通过参数对话框设置 algebraic constraint 的代数状态初始值，但是这时，建立代数环解方程就必须使用 algebraic constraint 模块；也可以通

过在连线上放置一个初始信号设置模块——IC 模块，来说明该连线代表的信号的初始值。

最后再强调一下，使用代数环时，应使用 IC 模块或者 algebraic constraint 模块来设置代数状态的初始猜测值。

### 5.2.2 非代数直接馈入环路

前面一节说过，一般情况下，包含直接馈入的环路都是代数环，这个通用法则也存在着例外：

- (1) 环路包含触发子系统。
- (2) 环路是一个模块的输出端口到积分器的重置端口。

在触发子系统的情况下，一个解法器假定子系统的输入信号在触发的时刻保持稳定。这就允许子系统使用前一个仿真时刻的输出结果来计算当前时间步的输入，这就避免了使用代数环解法器。

为了理解这一点，读者请看下面的示例模型。

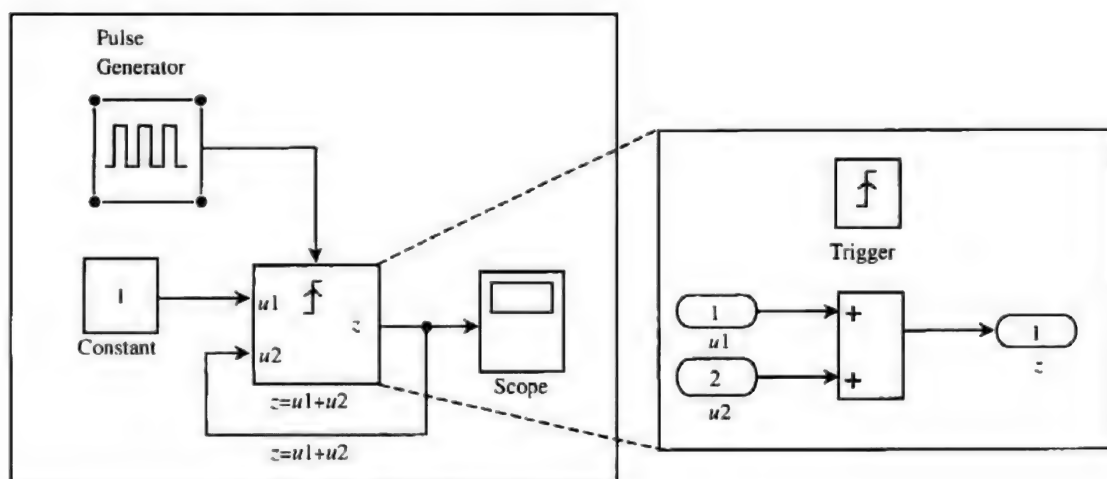


图 5-8 非代数环示例

在示例模型中，我们建立了一个简单的触发子系统  $z = u1 + u2$ ，图 5-8 的左图显示了它的内部图表。运行仿真后，得到的仿真结果如图 5-9 所示。



图 5-9 得到的仿真结果



读者可以看到，得到的结果并非像前面代数环一样，是一个方程的解；而是在每一个触发事件发生时，都把子系统的前一时刻的输出增加 1，于是 scope 的波形就是一个阶梯了。也就是计算

$$z = 1 + u$$

其中， $u$  是子系统前一时刻的输出，因为触发子系统的输出端口具有保持作用。

如果把触发模块去掉，模型就变成了图 5-10 所示的样子。

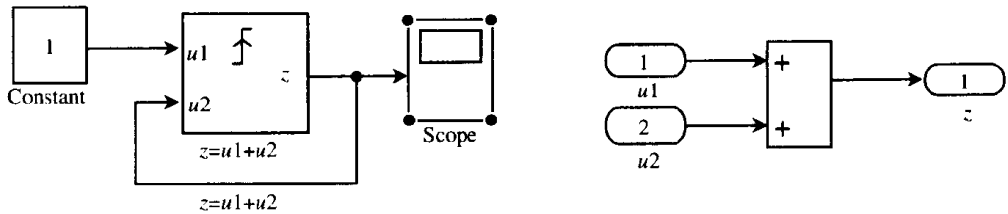


图 5-10 去掉触发模块的模型

在上图中，整个模型就是

$$1 + z = z$$

这是一个没有解的方程。

### 5.3 离散时间系统

Simulink 具有对离散（采样数据）系统进行仿真的能力，模型可以是多速率的——模型包含的模块具有不同的采样速率，还可以是混合系统——既有离散模块，又有连续模块。

Simulink 里的每一个离散模块在输入端口都有一个内置的采样器，和一个零阶保持器在输出端。当离散模块和连续模块混合在一起时，在两次采样时刻间，离散模块的输出保持一个常数。离散模块的输出只有在下一个采样时刻到来时，才会被更新。

在每一个离散系统的参数对话框中，都有一个 sample time 参数让用户设置模块的输出被更新的时刻。正常情况下，sample time 应该被设成标量，但 Simulink 也允许用户通过给 sample time 参数设置一个两个元素的向量来说明一个偏移时间。具体的方法是在 sample time 编辑框里输入[ts, offset]，ts 是指两次采样的时间间隔，而 offset 则指偏移时间。离散模块被更新的时间满足下面的关系：

$$t = n * ts + offset$$

其中， $n$  是一个整数，而 offset 则是绝对值比 ts 小的数，即正负均可。Offset 在一个模块必须比别的模块早一些或者晚一些更新时非常有用。

注意，在仿真运行时，是无法改变一个离散模块的采样时间的，这和模块的其他参数不同，如果要改变 sample time，必须先停止仿真。

### 1. 纯离散系统

纯离散系统可以用任何一种解法器来进行仿真，它们所求到的结果没有什么区别。

### 2. 多速率系统

多速率系统是指系统里包含以不同的速率采样的模块。它们既可以是纯离散系统，也可以是连续和离散的混合系统。图 5-11 就是一个多速率系统的例子。

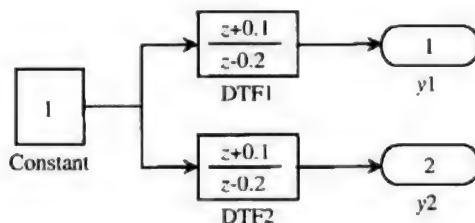


图 5-11 多速率系统示例

模型用到了一个 constant 模块、两个 discrete transfer fcn 模块以及两个 out 端口模块，其中 discrete transfer fcn 的位置在 math 子库。它们的传递函数都设置为：numerator:[1 0.1]，denominator:[1 -0.2]（见图 5-12）。

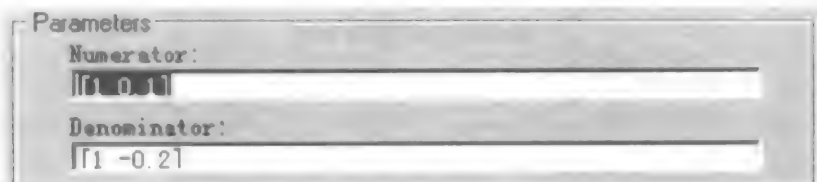


图 5-12 传递函数的设置

至于 sample time 参数的设置，DFF1 为[1 0.1]，而 DFF2 则为[1 -0.2]。

读者一定注意到了，模型中没有接收器类型的模块，而只有两个输出端口，这里用到了一项技术，Simulink 通过这两个端口把结果返回到 MATLAB 工作空间里的变量中。请读者在 MATLAB 命令窗口输入下面的命令。

```
>> [t, x, y] = sim('multirate', 5); % 通过命令行命令 sim 执行仿真，其中 multirate 是
                                     % 要运行的系统的名称，而后面的参数是指定仿真的时间
>> stairs(t, y); %画出输出的阶梯图
```

画出的曲线，如图 5-13 所示。从图中可以看出：

（1）离散系统的输出都是柱状图形，也就是说在采样点之间，模块的输出保持不变。

（2）由于 DFF1 和 DFF2 的 sample time 参数设置不同，所以它们发生跳变的时刻不同，在 y2 曲线上明显可以看出一个偏移，它的第一次跳变从大于 0 的时刻开始，而 y1 则从 0 时刻开始。

### 3. 用颜色来反映采样时间

在 Simulink 的 format 菜单下有一个 sample rate colors 选项，它的作用就是用不同的颜色来反映模型中不同模块的采样速率的快慢。表 5-1 是不同的颜色所表示的意义。

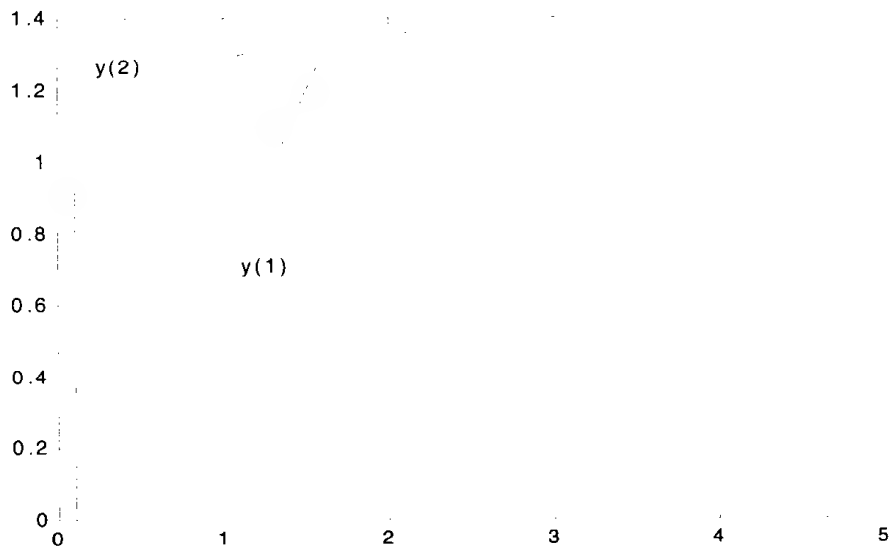


图 5-13 多速率系统的输出曲线

表 5-1 采样时间的颜色表示

颜 色	表示的意义
黑色	连续模块
红紫色	常数模块
红色	最快的离散采样时间
绿色	次最快的离散采样时间
蓝色	第三快的离散采样时间
亮兰	第四快的离散采样时间
暗绿	第五快的离散采样时间
兰绿	受触发的离散采样时间
灰色	固定的最小的采样时间
黄色	混合模块（组合模块的子系统或者是组合信号的 demux 和 mux 模块，它们的组成元素有不同的采样速率）

请读者注意，就像上表所示的一样，Simulink 为各个模块设置的颜色不是指绝对的采样时间，而是指它相对于其它模块的采样时间的快慢。为了让读者更好的理解 Simulink 的这个特性，下面我们来介绍 Simulink 的采样时间传播引擎（STPE）。图 5-14 演示了一个采样时间为  $T_s$  的 Discrete Filter 模块驱动一个增益模块的情形。因为增益模块的输出仅仅是简单地把输入乘以一个常数，所以它的输出和离散滤波模块的输出以相同的速率更新，也就是说增益模块的有效采样速率等于滤波器的采样速率。这就是隐藏在 STPE 后面的基本机制。

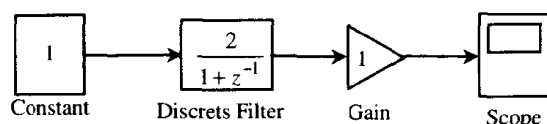


图 5-14 采样时间

建好模型后，就可以用 **format** 菜单下的 **sample time feature** 特性来看看采样时间的颜色特性。上面的这个模型比较简单，读者可以用 **Simulink** 的演示模型来试验这个特性，会看得更清楚些。在模型改变时，**Simulink** 并不会自动地去更新各模块的采样时间颜色，而是需要用户用 **Edit** 菜单下的 **update the diagram** 命令去更新。

**Simulink** 遵照以下法则为单个的模块设置采样速率：

(1) 连续模块（例如积分模块，微分模块，传递函数模块等等），按定义，采样时间为连续的。

(2) 常数模块（例如，常数模块），按定义，采样时间是常数。

(3) 离散模块（如零阶保持模块，单位时延模块，离散传递函数模块），它们的采样时间由用户在模块对话框的显式说明来决定。

(4) 其他的模块，它们的采样时间取决于输入信号的采样时间。比方说，一个 **Gain** 模块当它由一个积分模块来驱动时，就是一个连续模块；而在由 **Zero-OrderHold** 模块驱动时，却被当成和一个 **Zero-OrderHold** 模块相同的采样时间的离散模块。

(5) 具有不同的采样时间的输入信号的模块，如果所有的采样时间都是最快采样时间的整数倍，那么整个模型的仿真时间都被设置成最快的采样时间。这一点很容易理解，**Simulink** 就可以保证每个输入信号所需要的采样时间点都能被产生。而在模块内部，**Simulink** 检查当前的仿真时间步是否和模块定义的采样时间重合，如果重合就更新模块的输出。如果可变步长解法器被使用，模块的采样时间就是连续时间。如果定长解法器被使用，而且所有信号的采样时间的最大公因子——基准时间——能够被计算，那么这就是模块的采样时间。

但读者要注意到 **demux** 和 **mux** 模块是组合信号的虚模块——信号保留原来的定时信息通过该模块。因此，如果 **demux** 模块引出的直线是具有不同的采样时间的源驱动，那这些直线就会有不同的颜色。而 **mux** 和 **demux** 模块本身被赋成黄色，表示它们是混合的，具有多个速率的信号。

类似的，对于包含不同采样速率模块的子系统，也用黄色来表示，只有在子系统内的每一个模块都在相同的采样速率工作时，**Simulink** 才会根据这个速率来确定子系统的颜色。

在某些情形下，**Simulink** 也能向后传递采样时间到源模块，如果这样做不会影响仿真的结果。例如，在下面的模型中（图 5-15），**Simulink** 识别出信号发生器模块驱动离散积分模块，所以 **gain** 模块和信号发生器模块的采样时间都被设成离散积分模块的采样时间。

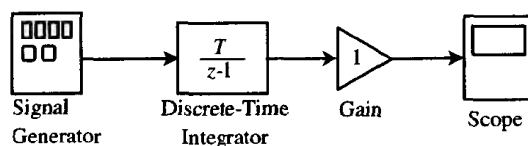


图 5-15 采样时间的传播

## 5.4 使用回调函数

### 5.4.1 回调函数基本概念

在具体讲解回调函数的用法和作用前,我们先来看看 MATLAB 中使用 `callback` 函数的一个演示程序。请读者在 MATLAB 命令窗口输入

```
>> f14;
```

f14 是 Simulink 提供的一个复杂的仿真模型示例,读者也可以通过在 `matlab demo` 窗口手动找到这个模型。这里提及这个模型的目的不是为了介绍这个模型的原理、具体的构造技巧,主要是让读者体会一下什么是回调函数,它能起什么作用。图 5-16 是 f14 的模型图表。

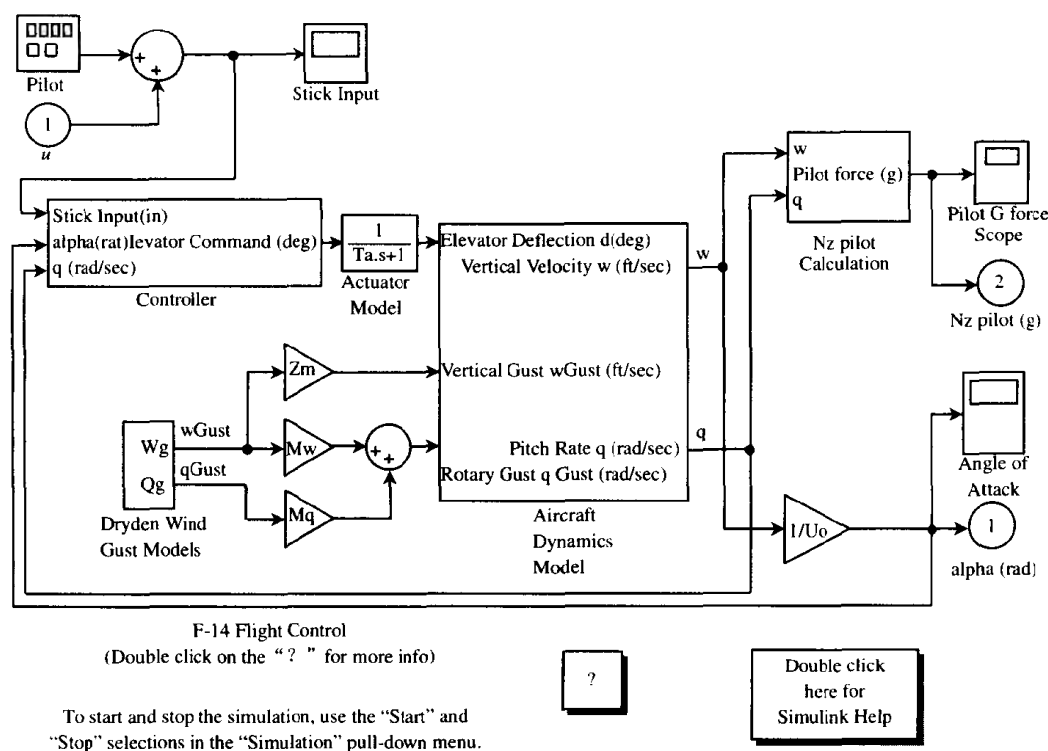


图 5-16 f14 模型图表

请读者先克服对 f14 模型的作用以及各个模块组成原理的好奇,把注意力放在本书下面要让读者思考的问题。在 f14 的许多模块的参数对话框里,经常出现一些变量,而在模型中却找不到任何关于这些变量的定义。关于在参数对话框里使用变量的情形,本书的第四章的“封装子系统”一节,在讲解参数传递的两种方式: `evaluate` 和 `literal` 时,就曾经举过例子,印象不深的读者可以回过头去温习一下。但在那里,设置模块参数为变量时,在运行前都要在 MATLAB 工作空间先定义这个变量。因为按照 `evaluate` 模式(这是 Simulink 模块常用的参数传递模式)的原理,任何在参数设置对话框输入的内容,首先被 MATLAB 进行估值,其实也就相当于把它们输入在 MATLAB 命令窗口,求出这些表达式的值,再传给模块。根

据 MATLAB 的估值顺序（见第二章），一个字符串首先被认为是 MATLAB 工作空间里的一个变量，如果不是，就认为是内置函数，依此类推。所以在参数设置时，使用已在 MATLAB 工作空间定义过的变量（其实可以使用一切可以估值的表达式）当然是可行的。但在 f14 模型里，似乎不需要这样的过程。模型被打开之后，无需在 MATLAB 里定义这些变量，至少表面上看来不需要，就可以顺利运行模型的仿真。

真的是不需要事先定义这些变量吗？其实不然，试想如果没有定义，那 MATLAB 又从何估值呢？请读者在 MATLAB 命令窗口输入 who 命令查询变量。

```
>> who % 查询变量
```

```
Your variables are:
```

Ka	Mq	Tal	W2	b
Kf	Mw	Ts	Wa	beta
Ki	Sa	Uo	Zd	cmdgain
Kq	Swg	Vto	Zw	g
Md	Ta	Wl	a	gamma

也就是说，f14 中用到的一些变量其实都已经事先定义了，只是定义的方式不是以前的那种手工定义，而是通过自动调用执行某些 MATLAB 命令和函数来完成。读者可以预计的方式在模型被打开时，调用某个程序或函数。这种预期和客观的技术事实是基本吻合的，f14 这种功能的实现，依赖于 callback 函数（回调函数）。

所谓回调函数，是指当用户定义的模型的图表或者模块发生某种特殊行为时，执行的 MATLAB 表达式（M 文件和 M 文件函数）。例如，在 f14 模型打开时，Simulink 就按照在模型建立时设置好的回调函数来执行定义变量的工作。回调函数有点像许多高级编程语言里的事件处理程序。在 MATLAB 里，为模型或模块的某种行为设置回调函数的方法是，设置与该行为对应的参数为相应的回调函数名，这些参数就像高级程序语言里的事件。例如，要在模型被载入前，执行一段用于定义一些数据变量参数（如同 f14 模型）的 M 文件，就可以设置模型的 PreLoadFcn 参数为该 M 文件的文件名称。在 Simulink 里，为模型设置参数的方法是使用 set\_param 命令。例如，要设置 PreLoadFcn 参数为 ‘modeldat’ 这一 M 文件名，就可以使用下面的命令

```
>> set_param('模型名称','PreLoadFcn','modeldat');
```

在举例说明如何使用 set\_param 命令来设置回调方法参数之前，我们来检验一下 f14 模型的回调函数。f14 模型的 PreLoadFcn 回调方法参数真的被设置了一个回调函数吗？在 Simulink 里可以用 get\_param 命令来获取模型文件中某个参数的值。例如，这里可以在 MATLAB 命令窗口输入

```
>> get_param('f14','PreLoadFcn')
```

```
ans =
```

```
f14dat
```

命令执行后，f14 模型的 PreLoadFcn 参数被设置了一个名为 f14dat 的 M 文件（函数）。不难查找到这个 M 文件，它的位置是你的 MATLAB 安装目录下的 toolbox \Simulink \simdemos 目录。整个 M 文件的代码如下。

```
% Numerical data for F-14 demo
%   Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
%   $Revision: 1.9 $
g = 32.2;
Uo = 689.4000;
Vto = 690.4000;
% Stability derivatives
Mw = -0.00592;
Mq = -0.6571;
Md = -6.8847;
Zd = -63.9979;
Zw = -0.6385;
% Gains
cmdgain = 3.490954472728077e-02;
Ka = 0.6770;
Kq = 0.8156;
Kf = -1.7460;
Ki = -3.8640;
% Other constants
a = 2.5348;
gamma = 0.0100;
b = 64.1300;
beta = 426.4352;
Sa = 0.005236;
Swg = 3;
Ta = 0.0500;
Tal = 0.3959;
Ts = 0.1000;
W1 = 2.9710;
W2 = 4.1440;
Wa = 10;
```

里面的命令定义了 f14 里用到的一些变量。于是，在 Simulink 载入这个模型之前，就会先调用 f14dat 这个 M 文件，在 MATLAB 工作空间里定义好用到的变量。这个参数设置的相

应命令就是

```
>> set_param('f14', 'PreLoadFcn', 'f14dat');
```

但是建议读者不要对 Simulink 的示例模型的参数进行随便的试验，即使是修改，也请先对模型进行备份。比较好的方法是建立一个简单模型来试验它的用法。

至此，前面提到的 f14 模型中出现的疑问得到比较完满的解释。

剩下的一个关键问题是，模型或模块中有哪些参数用于定义回调方法，而这些方法又在什么时候调用。表 5-2 列出了这些回调参数以及对应的回调方法的执行时机。

**表 5-2** 模型的回调参数

回调参数名称	回调方法执行的时机
CloseFcn	在模块图表被关闭之前
PostLoadFcn	在模型被载入之后，为这个参数定义一个回调方法对产生一个要求模型已经存在的界面非常有用
InitFcn	在模型的仿真开始时调用
PostSaveFcn	在模型被保存之后
PreLoadFcn	在模型被载入之前，用于载入预先载入模型使用的变量
PreSaveFcn	在模型被保存之前
StartFcn	在模型仿真开始前
StopFcn	在模型仿真停止之后，在 StopFcn 执行前，仿真结果先被写入工作空间中变量和文件中
CloseFcn	当模块是使用 close_system 命令关闭时
CopyFcn	在模块被拷贝之后，这个回调对子系统是递归的（意思是，当用户复制一个子系统，它包含 CopyFcn 参数被定义过的模块，那么回调方法也执行）。如果使用 add_block 命令复制模块，这个方法也被执行
DeleteFcn	在模块被删除之前，这个回调对子系统递归
DestroyFcn	当模块被毁坏时
InitFcn	在模块图表被编译和模块参数被估值之前
LoadFcn	当模块图表被载入，该回调对子系统递归
ModelCloseFcn	在模块图表被关闭之前，该回调对子系统递归
MoveFcn	当模块被移动或者调整大小时
NameChangeFcn	当模块的名称或者路径改变后，当子系统的路径被改变时，它递归的为子系统所包含的每个模块在调用它们本身的 NameChangeFcn 方法后，调用该方法
OpenFcn	当模块被打开时。这个参数一般用于子系统模块。该方法在用户双击打开模块，或者使用以该模块作为参数的 open_system 命令时，被调用。OpenFcn 参数重载打开模块一般发生的行为，例如，显示对话框或者打开模块
ParentCloseFcn	在关闭包含该模块的子系统前，或者在该模块组成用 new_system 新建的子系统的一部分时
PreSaveFcn	在模块图表被保存前。该回调对于子系统模块是回调的
PostSaveFcn	在模块图表被保存后。该回调对于子系统模块是回调的
StartFcn	模块图表被编译之后，仿真开始之前
StopFcn	在仿真的以任何一种形式结束时
UndoDeleteFcn	当一个模块删除操作被取消时



下面就来举几个简单的示例模型来说明什么时候以及如何设置回调方法。

### 5.4.2 回调函数示例

这里将用两个例子来说明回调函数的使用。

#### 1. 对变量进行初始化

这一个例子基本上就是对前面 f14 模型的一个简单回顾，它使用一个回调函数 'exam\_542dat' 来对模型 exam\_542 进行变量初始化。而在模型结束时将仿真结果绘图。

请读者先按图 5-17 所示的图标建立好模型，其中 gain 模块的参数设置为 gain，请在模型仿真参数对话框的 Workspace I/O 页设置，把时间和输出结果保存到工作空间。

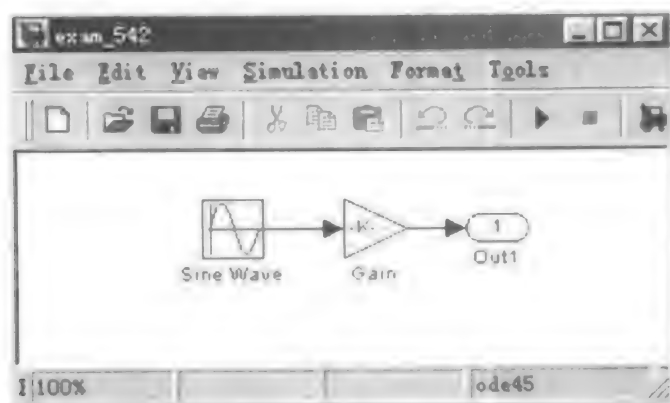


图 5-17 模型图表

然后，请读者在 MATLAB 命令窗口输入

```
>> set_param('exam_542', 'PreLoadFcn', 'ini_gain');
```

```
>> set_param('exam_542', 'StopFcn', 'out_graphic');
```

之后，保存模型，并关闭模型。打开 MATLAB 编辑器，新建一个名为 ini\_gain 的 M 文件，并在里面输入

```
gain=2;
```

再新建一个名为 out\_graphic 的 M 文件，里面的语句为

```
plot(tout, yout);
```

保存这些 M 文件之后，打开 exam\_542 模型并且运行它，读者可以发现，仿真结束之后，Simulink 会自动调用回调函数对保存在工作空间的输出变量进行处理，这里是用 plot 命令绘出模型输出的波形。

#### 2. 建立动态对话框

这里用一个简单的例子，演示如何建立动态对话框。关于它的基本原理在第四章已经介绍过，这里就不再重复。

请读者新建一个模型，并把一个 SubSystem 模块复制到模型中，请按照图 5-18 所示的对话框对子系统模块进行封装。封装对话框的示意图为图 5-19。

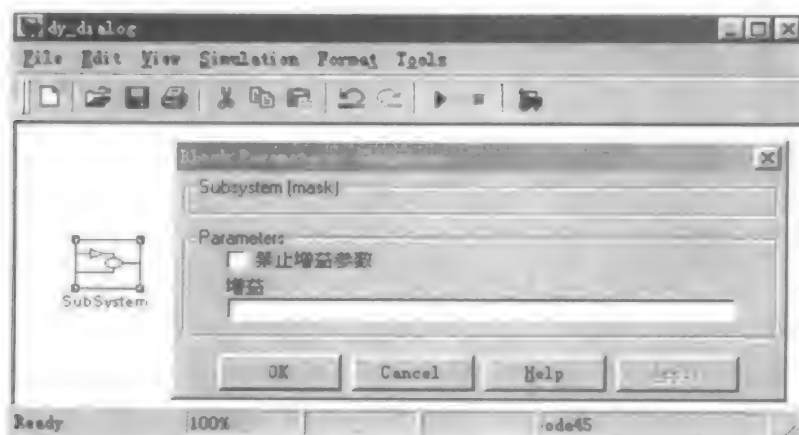


图 5-18 动态对话框

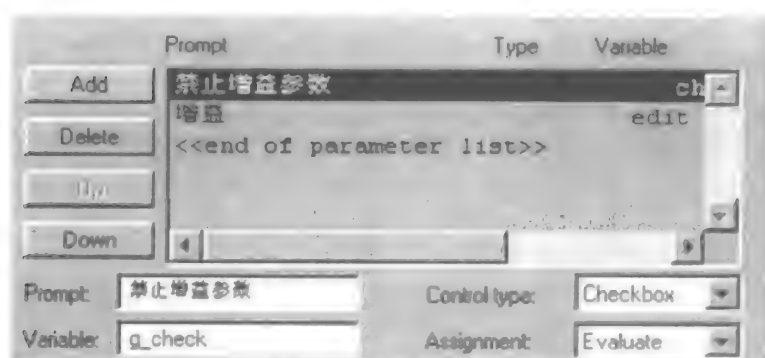


图 5-19 封装对话框设置

表 5-3 给出了其中各模块参数的具体定义。

**表 5-3 各模块参数的具体定义**

Prompt	variable	Control type	Assignment
禁止增益参数	g_check	Checkbox	Evaluate
增益	gain	Edit	Evaluate

这个例子要达到的效果是，当“禁止增益参数”检查框被选中之后，增益编辑框应该不能输入，也就是处于禁止状态，呈现为灰色。下面就通过回调函数来实现这个功能。

为此，首先要为检查框控件定义回调函数。具体的操作，先在模型里选择子系统模块，然后在 MATLAB 工作空间输入下面的命令

```
>>set_param(gcb,'MaskCallbacks',{'disable_gain','});
```

这个命令通过设置当前模块（子系统模块）的第一个参数的回调函数为 `disable_gain`，别忘了保存模型。于是当第一个参数被改变时，Simulink 会自动调用这个 `dsable_gain.m` 文件。因此只要在 `dsable_gain.m` 里写入控制“增益”编辑框的语句即可。下面列出了 `dsable_gain.m` 的代码。

```
m_value=get_param(gcb,'MaskValues'); % 获得个参数控件的当前值
```

```

if strcmp(char(m_value(1)), 'on')           % 检查框的值是'on'吗
    set_param(gcb, 'MaskEnables', {'on','off'}); % 是，将第二个参数控件禁止
else
    set_param(gcb, 'MaskEnables', {'on','on'}); % 否，将第二个参数控件使能
end

```

至于上面代码中的各个封装对话框参数的作用，请读者参照第四章。这里只想解释一下 if 判断语句。因为 `m_value` 返回的是一个 `cell` 类，它不支持直接进行比较，所以先使用 `char` 函数将其转化为字符串，然后再用 `strcmp` 命令比较它是否和 `on` 相同。也许有的读者还会问，对于 `checkbox` 和它不是有一个变量关联，那为什么不用它来进行判断，直接是整数就不用这么麻烦。问题在于那里面的关联变量是在封装子空间定义的，而在 **MATLAB** 工作空间自然就无法访问了。

至此，整个动态对话框就完成了，完成后的动态对话框的效果如图 5-20 所示。

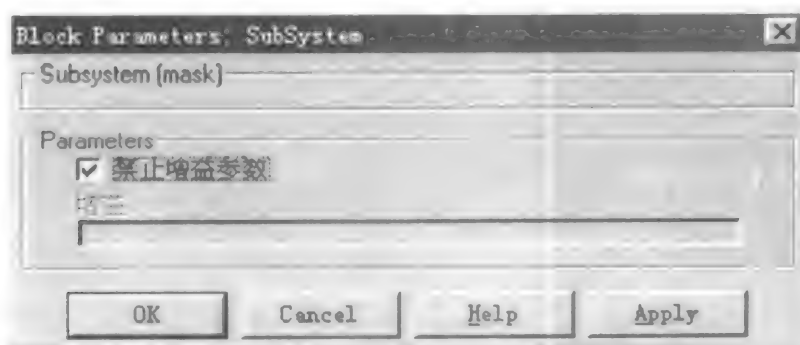


图 5-20 动态对话框效果图

## 5.5 模型文件格式

**Simulink** 为用户提供了图形界面和命令行两种方式来建立模型，设置模型的参数。这些方法，在某些场合给人的感觉是不太方便的。实际上，**Simulink** 允许用户直接修改模型文件，当然这要建立在对其的足够了解的基础上。这一节就粗略介绍 **Simulink** 的模型文件的格式。

在 **Simulink** 里每一个模型（包括库）都保存在一种以 `.mdl` 为后缀名的文件里，称为模型文件。概括地说，一个模型文件是一个结构化了的 ASCII 文件，它包括描述模型的关键字和参数——取值对。读者可以通过 **MATLAB Edit** 来查看模型文件，方法和打开一般的 `M` 文件没什么区别，可以用 **File** 菜单下的 `open` 命令，但打开时要把打开对话框的文件类型变成所有文件，否则就看不到当前目录中的模型文件。图 5-21 是一个模型文件（**Simulink** 的示例模型 `vdp`，其位置是在读者的安装目录下 `toolbox\Simulink\simdemos`），显示在 **MATLAB edit** 中。



```

...
Branch {
  <Branch Parameter Name> <Branch Parameter Value>
  ...
}
}
Annotation {
  <Annotation Parameter Name> <Annotation Parameter Value>
  ...
}
}
}

```

读者可以对照这个基本结构来浏览 vdp 模型的模型文件。

Simulink 的模型文件是由描述模型的不同组成部分的 section 构成的，它们分别是：

- (1) 定义模型参数的 Model section;
- (2) 包含模型中的模块的缺省设置的 BlockDefaults section;
- (3) 包含模型中注解的缺省设置的 AnnotationDefaults section;
- (4) 包含模型中的每一个系统（包括顶层系统和每一个子系统）的描述参数的 System section。每一个 System section 包含 block、line 和 annotation 等描述。

下面就来具体看看这些 section。为了说明方便，我们用第四章的封装子系统一节中图 (5-22)  $mx+b$  模型的模型文件为例来说明各个 section。

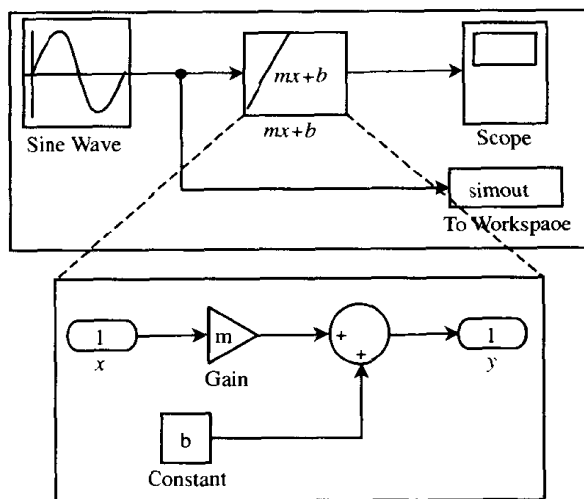


图 5-22  $mx+b$  子系统

(1) **Model section**。Model section 位于模型文件的顶部，它定义了模型层次参数的取值。这些参数包括模型名称、模型版本和仿真参数。读者可以在下面的示例模型文件的 Model section 里找到在第五章里介绍的，用仿真参数对话框设置的各种仿真参数。也就是说，读者完全可以从模型文件设置参数。

例如，下面代码所示的原来的模型文件中，LimitMaxRows 和 MaxRows 这两个参数的值分别是 off 和"1000"。如果读者把它们分别改为 on 和"2000"，保存模型文件后（注意如果已经打开了该模型图表，请先关闭，关闭时请不要保存文件，否则会覆盖了在 matlab edit 中做的修改），再在 Simulink 里打开模型。查看该模型的仿真参数，读者会发现在对话框里的相应参数也作了修改（图 5-23）。

☒ Limit rows to last: 2000

图 5-23 用模型文件来修改仿真参数

```
Model {
  Name          "sampleformask"
  Version       3.00
  SimParamPage  "Solver"
  SampleTimeColors off
  InvariantConstants off
  WideVectorLines off
  ShowLineWidths off
  ShowPortDataTypes off
  StartTime     "0.0"
  StopTime      "10.0"
  SolverMode     "Auto"
  Solver        "ode45"
  RelTol        "1e-3"
  AbsTol        "auto"
  Refine        "1"
  MaxStep       "auto"
  InitialStep   "auto"
  FixedStep     "auto"
  MaxOrder      5
  OutputOption  "RefineOutputTimes"
  OutputTimes   "[]"
  LoadExternalInput off
  ExternalInput "[t, u]"
  SaveTime      on
  TimeSaveName  "tout"
  SaveState     off
  StateSaveName "xout"
  SaveOutput    on
  OutputSaveName "yout"
}
```

LoadInitialState	off
InitialState	"xInitial"
SaveFinalState	off
FinalStateName	"xFinal"
SaveFormat	"Matrix"
LimitMaxRows	off
MaxRows	"1000"
Decimation	"1"
AlgebraicLoopMsg	"warning"
MinStepSizeMsg	"warning"
UnconnectedInputMsg	"warning"
UnconnectedOutputMsg	"warning"
UnconnectedLineMsg	"warning"
InheritedTsInSrcMsg	"warning"
IntegerOverflowMsg	"warning"
UnnecessaryDatatypeConvMsg	"none"
Int32ToFloatConvMsg	"warning"
SignalLabelMismatchMsg	"none"
ConsistencyChecking	"off"
ZeroCross	on
SimulationMode	"normal"
BlockDataTips	on
BlockParametersDataTip	off
BlockAttributesDataTip	off
BlockPortWidthsDataTip	off
BlockDescriptionStringDataTip	off
BlockMaskParametersDataTip	off
ToolBar	off
StatusBar	off
BrowserShowLibraryLinks	off
BrowserLookUnderMasks	off
OptimizeBlockIOStorage	on
BufferReuse	on
BooleanDataType	off
RTWSystemTargetFile	"grt.tlc"
RTWInlineParameters	off
RTWRetainRTWFile	off
RTWTemplateMakefile	"grt_default_tmf"
RTWMakeCommand	"make_rtw"

```

RTWGenerateCodeOnly    off
ExtModeMexFile         "ext_comm"
ExtModeBatchMode       off
ExtModeTrigType        "manual"
ExtModeTrigMode        "oneshot"
ExtModeTrigPort        "1"
ExtModeTrigElement     "any"
ExtModeTrigDuration    1000
ExtModeTrigHoldOff     0
ExtModeTrigDelay       0
ExtModeTrigDirection   "rising"
ExtModeTrigLevel       0
ExtModeArchiveMode     "off"
ExtModeAutoIncOneShot  off
ExtModeIncDirWhenArm   off
ExtModeAddSuffixToVar  off
ExtModeWriteAllDataToWs off
ExtModeArmWhenConnect  off
Created                "Tue Dec 12 21:40:29 2000"
UpdateHistory          "UpdateHistoryNever"
ModifiedByFormat       "%<Auto>"
ModifiedDateFormat     "%<Auto>"
LastModifiedDate       "Fri Dec 29 16:19:33 2000"
ModelVersionFormat     "1.%<AutoIncrement:7>"
ConfigurationManager   "none"
}

```

(2) BlockDefaults section。BlockDefaults section 出现在模型文件的仿真参数后面，它定义了这个模型内的模块参数的缺省值。这些值可以被 block section 里定义的单个模块参数重定义。

```

BlockDefaults {
    Orientation        "right"
    ForegroundColor    "black"
    BackgroundColor    "white"
    DropShadow         off
    NamePlacement      "normal"
    FontName           "Helvetica"
    FontSize           10
    FontWeight         "normal"
}

```



```
    FontAngle      "normal"
    ShowName       on
}
```

(3) AnnotationDefaults section。它出现在 BlockDefaults section 后面，定义了模型中所有注释的参数的缺省值。这些参数值不能用 set\_param 命令来修改。

```
AnnotationDefaults {
    HorizontalAlignment    "center"
    VerticalAlignment      "middle"
    ForegroundColor        "black"
    BackgroundColor        "white"
    DropShadow             off
    FontName               "Helvetica"
    FontSize               10
    FontWeight             "normal"
    FontAngle              "normal"
}
LineDefaults {
    FontName               "Helvetica"
    FontSize               9
    FontWeight             "normal"
    FontAngle              "normal"
}
```

(4) System section。顶层的系统 and 模型中的每一个子系统都在一个单独的 System section 里描述。每一个 System section 部分定义了系统级的参数，并包含系统内每一个模块的 Block、Line 和 Annotation 节，以及系统自身的 Line 和 Annotation 小节。具有分枝点的直线还包含一个 Branch section 来定义分枝直线。

这里，请读者仔细体会模型文件体现出的模型结构的层次性。在 System section 里有四个 Block section，分别对应四个顶层模块：scope、sin、toWorkspace 和子系统模块  $mx+b$ 。而子系统  $mx+b$  对应的 Block section 又嵌套了一个 System section，用以说明子系统内部的模块和连线的参数。

```
System {
    Name              "sampleformask"
    Location           [160, 123, 500, 260]
    Open              on
    ModelBrowserVisibility off
    ModelBrowserWidth 200
    ScreenColor        "automatic"
```

```

PaperOrientation      "landscape"
PaperPositionMode     "auto"
PaperType             "usletter"
PaperUnits            "inches"
ZoomFactor            "100"
AutoZoom              on
ReportName            "Simulink-default.rpt"
Block {
    BlockType          Scope
    Name               "Scope"
    Ports              [1, 0, 0, 0, 0]
    Position           [240, 19, 270, 51]
    Floating           off
    Location           [305, 135, 629, 375]
    Open              off
    NumInputPorts      "1"
    TickLabels         "OneTimeTick"
    ZoomMode           "on"
    List {
        ListType       AxesTitles
        axes1          "%<SignalLabel>"
    }
    Grid               "on"
    TimeRange          "auto"
    YMin               "-5"
    YMax               "5"
    SaveToWorkspace    off
    SaveName           "ScopeData"
    DataFormat         "StructureWithTime"
    LimitMaxRows       on
    MaxRows            "5000"
    Decimation         "1"
    SampleInput        off
    SampleTime         "0"
}
Block {
    BlockType          Sin
    Name               "Sine Wave"
    Position           [50, 20, 80, 50]

```



Location [282, 214, 565, 342]

Open off

ModelBrowserVisibility off

ModelBrowserWidth 200

ScreenColor "white"

PaperOrientation "landscape"

PaperPositionMode "auto"

PaperType "usletter"

PaperUnits "inches"

ZoomFactor "100"

AutoZoom on

Block {

BlockType Inport

Name "x"

Position [15, 23, 45, 37]

Port "1"

PortWidth "-1"

SampleTime "-1"

DataType "auto"

SignalType "auto"

Interpolate on

}

Block {

BlockType Constant

Name "Constant"

Position [45, 65, 75, 95]

Value "b"

}

Block {

BlockType Gain

Name "Gain"

Position [80, 15, 110, 45]

Gain "m"

SaturateOnIntegerOverflow on

}

Block {

BlockType Sum

Name "Sum"

Ports [2, 1, 0, 0, 0]

Position [140, 20, 160, 40]  
ShowName off  
IconShape "round"  
Inputs "|++"  
SaturateOnIntegerOverflow on  
}

Block {  
BlockType Outport  
Name "y"  
Position [200, 23, 230, 37]  
Port "1"  
OutputWhenDisabled "hold"

```
    }  
  }  
  Line {  
    SrcBlock      "Sine Wave"  
    SrcPort       1  
    Points        [20, 0]  
    Branch {  
      DstBlock     "mx+b"  
      DstPort      1  
    }  
    Branch {  
      Points       [0, 65]  
      DstBlock     "To Workspace"  
      DstPort      1  
    }  
  }  
  Line {  
    SrcBlock      "mx+b"  
    SrcPort       1  
    DstBlock      "Scope"  
    DstPort       1  
  }  
}  
}
```



## 第六章 仿真运行和结果分析

建好一个模型之后，下面的事情就是运行模型和分析仿真结果。前面的章节，对于运行仿真，只是粗略地提了一下，这一章将对它进行详尽地讲解。

### 6.1 使用菜单命令运行仿真

使用菜单命令运行仿真十分简便，而且用户和模型间交互能力强。这些命令允许用户选择常微分方程（ODE）解法和改变仿真的参数，而无需去记忆 MATLAB 的命令语法。更为重要的是，这种方法使得用户在仿真运行中可以进行某些操作，它们是：

- （1）可以修改许多仿真参数，包括仿真结束时间、解法和最大步长；
- （2）可以同时仿真其他系统；
- （3）可以单击模型的连线，通过浮动的 scope 或 display 模块来查看该直线传递的信号取值；
- （4）可以修改模块的参数，只要这种修改不会导致下面一些量的变化：模块状态、输入或输出的数目，采样时间，过零点的数目，模块参数的向量长度，内部模块工作向量的长度。

在仿真运行中，Simulink 不允许对模型结构进行任何修改，诸如添加或删除一个模块、一条连线等等。要进行这些修改，必须先终止仿真，修改完后，再运行仿真。

用菜单命令运行仿真的方法，其实在前面的学习中读者已经接触到了，例如，从 simulation 菜单下选择 start 命令，或者使用工具栏上的运行快捷按钮。但前面所用到的那些操作都是很不完整的，运行一个仿真完整的过程应该要分成三个步骤：设置仿真参数，开始仿真和查看结果。

#### 1. 设置仿真参数和选择解法器

设置仿真参数和选择解法器，选择 simulation 菜单下的 Parameters 命令。选择这个命令之后，Simulink 会显示一个仿真参数对话框，它用三个页面来管理仿真的参数：

- （1）solver 页，它允许用户设置仿真的开始和结束时间，选择解法器，说明解法器参数，以及选择一些输出选项；
- （2）Workspace/I/O 页，作用是管理模型从 MATLAB 工作空间的输入和对它的输出；
- （3）Diagnostics 页，允许用户选择 Simulink 在仿真中显示的警告信息的等级。

本章的下一节将会对这些仿真参数设置页面进行详细的说明。

仿真参数可以被设置为有效的 MATLAB 的表达式，包括常数、工作空间变量名、MATLAB 函数和数学运算符。

设置完参数后，就可以按下对话框下的 apply 按钮以应用你的设置，若要同时关闭按钮，



则选择 OK 按钮。如果是按下 Cancel 按钮，则会取消前面所作的任何修改。

## 2. 开始仿真

做好上面的工作之后才是开始仿真，除了我们已经用到的两种方法，还可以用快捷键 CTRL+T 来完成这个工作。提前结束仿真，同样可以在 simulation 菜单和工具栏找到相应的命令，又可以使用与开始仿真相同的快捷键。此外，Simulink 还允许用户暂停仿真，这些命令都在 simulation 菜单里面，它们在仿真运行后会自动出现，读者可以建立一个简单的模型来试验这些命令，在试验时，建议读者先把仿真时间设置长一点。

## 3. 仿真诊断对话框

如果仿真过程中有错误产生，Simulink 会中断仿真并在仿真诊断对话框里显示错误信息。

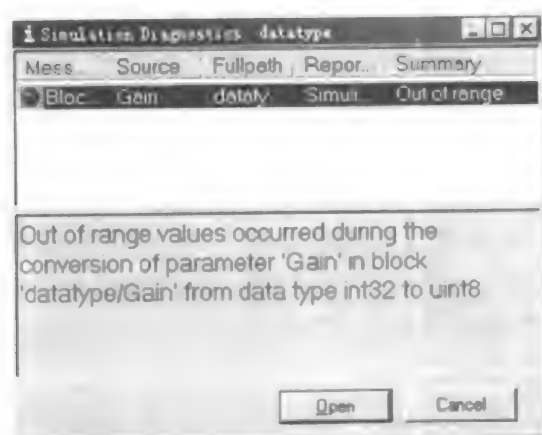


图 6-1 仿真诊断对话框

图 6-1 就是一个诊断对话框示例。它分为两个面板，上面的面板显示了每一个错误的信息。这些信息有：

- Message——消息类型（例如，模块错误、警告、log）；
- Source ——导致错误的模型元素（连线或模块等）的名称；
- FullPath——导致错误的元素的路径；
- ReportedBy——报告错误的组件（如 Simulink、rtw）；
- Summary——错误消息的简写，便于在列里显示。

而 Simulink 会在对话框下面的面板，显示错误消息的完整内容，注意如果具有多个错误消息，显示在下部面板的信息和上部面板被选中的错误消息相对应。Simulink 除了能显示这些信息，还可以打开模型的图表，并把错误单元（模块或者直线）用黄色来表示。用户还可以双击上部面板的错误消息，Simulink 就会显示出出现错误的具体位置，如上例中，就会显示出 Gain 模块的参数设置对话框。另外一种替代的方法是 open 命令。

## 6.2 仿真参数对话框

在 Simulink4.0 里，仿真参数对话框分为 Solver 页、Workspace I/O 页、Diagnostics 页和 Advanced 页。它和 Simulink3.0 的区别在于增加了一个 Advanced 页，其余的变化则不是太大。

### 6.2.1 Solver 页

在 solver 页（图 6-2）可以进行的设置有：选择仿真开始和结束时间；选择解法器，并设定它的参数；选择输出选项。

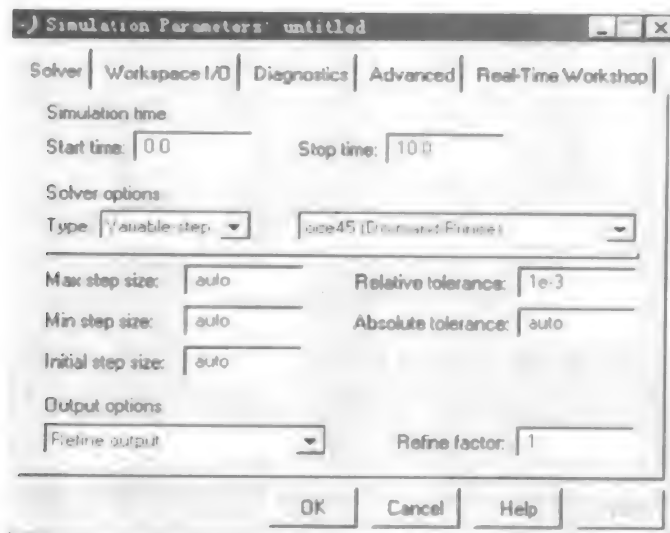


图 6-2 Solver 页

#### 1. 仿真时间

用户可以设置仿真的开始时间和结束时间。注意，这里的时间和真实时间并不一样，这里的时间只是计算机模拟中对时间的一种表示。比如 10 秒的时间，如果采样步长定为 0.1，则需执行 100 步，若把步长减小，则采样点数增加，那么实际的执行时间会增加。总的说来，执行一次仿真要耗费的时间依赖于很多因素，包括模型复杂度、解法器步长和计算机时钟速度。

#### 2. solver

Simulink 模型的仿真综合了常用的几种常微分方程，Simulink 提供了一些用于这些方程仿真的解法器。由于动态方程的差异性，所以有些解法器解决某些特定的问题会比另外一些解法器有效。因此，要获得精确而快速的仿真结果，必须仔细的选择解法器和设置它们的参数。

首先是根据仿真步长来进行选择，这时有两个选项：定长解法器和变长解法器。变长解法器可以在仿真过程中改变步长，这类解法器提供误差控制和过零检测。定长解法器在仿真过程中提供固定的步长，它们不进行误差控制和定位过零点。

##### （1）缺省解法器（Solver）

如果用户没有选择解法器，那么 Simulink 会根据你的模型是否有连续状态，自动选择一个。这可以分为两种情形：

（a）如果模型有连续状态，就选择 ode45，它是一种性能良好的通用解法器。然而，如果用户知道所仿真的模型是 stiff 问题，并且尝试 ode45 后不能获得可接受的结果，用户可以试试 ode15。

（b）如果模型没有连续状态，那么 Simulink 使用称为 discrete 的变长解法器，并且显示

一个消息表明模型现在使用的解法器不再是 ode45。注意 Simulink 除了提供一个称为 discrete 的变长解法器，也提供了一个称为 discrete 的定长解法器。

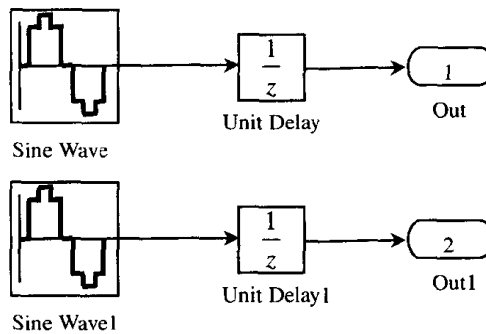


图 6-3 离散解法器的示例模型

为了理解它们之间的区别，请看图 6-3 所示的模型。其中，上面的 sine wave 模块的采样时间是 0.5 秒，而下面的 sine wave 的仿真时间是 0.75 秒，所以这个模型的基准采样时间是 0.25 秒（就是所有采样时间的最大公约数）。现在请读者，更改解法器的设置。首先是使用 variable-step 解法器，请在右边的列表框选择 discrete；设置完后，执行模型。注意到这个模型的 out 模块会把结果输出到 MATLAB 工作空间。读者可以用 who 查询 MATLAB 工作空间中存在的变量，会发现增加了两个变量 tout、yout。

```
>> tout
```

对于定长解法器，输出的仿真时间为：

```
tout = [0 0.2500 0.5000 0.7500 1.0000 1.2500 1.5000 1.7500 2.0000 2.2500
2.5000 2.7500 3.0000]
```

对于变长离散解法器，输出的仿真时间为：

```
[0.0 0.5 0.75 1.0 1.5 2.0 2.25 ...]
```

从上面的结果不难看出，定长离散解法器的步长是基准采样时间，而变长离散解法器采取允许的最大步长。

## （2）变长解法器

在 Simulink 参数对话框的 solver 页列出的变长解法器有：ode45, ode23, ode113, ode15s, ode23s 和 discrete；其中 ode45 是有状态的连续系统和无状态的离散系统的缺省设置。

（a）ode45 基于显式的 Runge-Kutta (4,5)（龙格-库塔）公式——Dormand-Princepair。它是单步解法器，也就是，在计算  $y(t_n)$  时，它仅需要最近处理时刻的结果  $y(t_{n-1})$ 。一般来说，面对一个仿真问题最好是首先试试 ode45。

（b）ode23 同样是基于 Bogacki 和 Shampine 的 Runge-Kutta (2,3) 的显式对。它在误差限要求不高和求解的问题不太难的情况下，可能会比 ode45 更有效。ode23 也是一个单步解法器。

（c）ode113 是一种阶数可变的 Adams-Bashforth-Moulton PECE 解法器。它在误差容许

要求严格的情况下通常比 ode45 有效。ode113 是一种多步解法器，也就是在计算当前时刻输出时，它需要以前多个时刻的解。

(d) ode15s 是一种基于数字微分公式的解法器 (NDFs)。它们和后向微分公式有联系但要比后者有效。和 ode113 一样，ode15s 也是一种多步解法器，如果用户估计要解决的问题是比较困难的，或者不能使用 ode45，或者即使使用效果也不好，就可以用 ode15s。

(e) ode23s 基于修改的二阶 Rosenbrock 公式。因为它是一个单步解法器，所以在弱误差允许下的效果好于 ode15s。它能解决某些 ode15s 所不能有效解决的 stiff 问题。

(f) ode23t 是梯形规则的一种自由插值实现。这种解法器适用于求解适度 stiff 的问题而用户又需要一个无数字振荡的解法器的情况。

(g) ode23tb 是 TR-BDF2 的一种实现，TR-BDF2 是具有两个阶段的隐式 Runge-Kutta 公式，它的第一个阶段是一个 trapezoidal 法则，第二个阶段是二阶的后向差分公式。在实现上，使用相同的循环矩阵来估算两个进程。

(h) discrete (变长)，当 Simulink 检查到模型没有连续状态时使用它。

- ✎ stiff 问题的解能以比积分间隔短得多的时间尺度变化，但实际上感兴趣的解往往以一个更长的时间尺度变化，而不是为 stiff 问题设计的解法。因为选用了足够解决最快变化的小时间步，所以在那些解变化缓慢的地方就没有什么效果了。

### (3) 定长解法器

可以选择的定长解法器包括 ode5、ode4、ode3、ode2、ode1 和 discrete，用法说明如下：

- (a) ode5 是 ode45 的固定步长版本，Dormand-Prince 公式；
- (b) ode4 是 RK4，四阶 Runge-Kutta (龙格—库塔) 公式；
- (c) ode3 是 ode3 的固定步长版本，Bogacki-Shampine 公式；
- (d) ode2 是 Heun 方法，也称为改进欧拉公式；
- (e) ode1 是欧拉方法；

(f) discrete (fixed-step) 是一个实现积分的定长解法器。它适合没有状态的模型，而且过零检测和错误控制对模型不重要。

如果觉得仿真的结果不满足工程要求，可以参阅本章的“提高仿真性能和精确性”一节。

### 3. 解法器选项

缺省的解法器选项对大多数问题都可以获得精确和有效的结果，但在有些情况，调整参数可能获得更好的结果。

### 4. Step size

对于变长解法器，用户可以设置最大的和推荐的初始步长参数，缺省情况下，步长自动的确定，它由值 Auto 表示。

对于定长解法器，用户可以设置固定步长，它的缺省值同样是 Auto。

#### (1) Maximum step size (最大步长参数)

最大步长参数决定了解法器能够使用的最大时间步长。它的缺省值由下面的公式确定：

$$h_{\max} = \frac{t_{\text{stop}} - t_{\text{start}}}{50}$$

一般来说，缺省值就足够了，如果用户担心漏掉关键的行为，那可以设置这个参数以防

止解法器采用过大的步长。如果仿真的时间跨度很长，那么缺省步长对解法器找到解而言就太长了。如果用户的模型具有周期性和类周期性的行为并且用户知道这个周期，那么就可以把最大的仿真步长设置为周期的一个系数（例如 1/4）。

### (2) Initial step size

初始步长参数，缺省情况下，解法器选择一个初始步长在仿真开始时刻检查状态的微分。如果第一个步长太大，解法器将会漏掉重要的行为。Initial step size 是建议的初始步长，解法器使用该步长，如果不能满足误差标准，就减小步长。

### (3) Minimum step size

设定解法器能够采取的最小时间步。如果解法器需要采用一个更小的仿真步来满足误差限度的要求，它会给出一个警告表示当前有效的相对误差限度。这个参数可以是一个大于零的实数值，也可以是一个两个元素的向量。向量的第一个元素表示最小步长的大小，而第二个元素表示 Simulink 在给出错误提示之前，最小步长误差提示所能出现的最大次数。比如，向量的第二个元素设为 0，表示在解法器第一次必须采取小于设定最小值的仿真步时，就产生错误。这等效于将 Diagnostics 页的最小步长冲突诊断错误选项置为 on 的状态。将第二个元素设置为-1，表示警告能够产生的次数没有限制。这个值也是当最小步长设置的值是标量时的缺省值。最小时间步参数元素的缺省值是机器的表示精度和-1（警告次数不限）。

## 5. Error Tolerances（误差限度）

解法器采用标准的当前误差技术来监控每一时间步的误差。在每一个时间步期间，解法器在该步结束时，计算状态值，并决定当前误差——这些状态值的估计误差。解法器然后把局部误差和可接受的误差相比较，后者是相对限度（rtol）和绝对限度（atol）的函数。如果有一个状态的当前误差大于可接收误差，解法器就会减小步长，并重新开始计算。

这里相对限度和绝对限度的区别是，相对限度是指误差相对于状态的值，是一个百分比。缺省值是 1e-3，表示状态的计算值要精确到 0.1%。而绝对误差限表示误差值的门限，或者说在状态值为零的情况下，可以接受的误差。

知道了相对限度和绝对限度，就可以按下面的公式计算出可接受的误差。

$$e_i \leq \max\{rtol * |x_i|, atol_i\}$$

如果 absolute tolerance 被设成了 auto，那么 Simulink 就为每一个状态设置初始 absolute tolerance 为 1e-6。当仿真进行时，Simulink 重新设置每个状态的 absolute tolerance 为最大的值。例如，如果一个状态从 0 变到 1，并且它的 rtol 为 1e-3，则在仿真的末尾，该状态的 atol 就会变成 1e-3。如果是从 0 变到 1000，相应的最终 atol 是 1。

这样计算得到的 atol 设置有可能不合适，那用户可以自己确定一个合适的值。最简单的方法，用户可以枚举不同 absolute tolerance 取值，分别用它们进行仿真，然后在其中确定一个结果最好的参数。

## 6. Mode（模式）

在用户选择了 fixed-step 解法器时，solver 页会显示一个 mode 列表，它里面的选项有：MultiTasking、SingleTasking 和 Auto。它们的意义分别如下：

(1) MultiTasking。选择这种模式时，当 Simulink 检测到模块间非法的采样速率转换，它会给出错误提示。所谓的非法采样速率转换指两个工作在不同采样速率的模块之间的直接

连接。在实时多任务系统中，如果任务之间存在非法采样速率转换，那么就有可能出现一个模块的输出在另一个模块需要时却无法利用的情况。通过检查这种转换，MultiTasking 将有助于用户建立一个符合现实的多任务系统的有效模型。

使用速率转换模块可以减少模型中的非法速率转换。Simulink 提供了两个这样的模块：unit delay 模块和 Zero-Order Hold 模块。对于从慢速率到快速率的非法转换，可以在慢输出端口和快输入端口插入一个 Unit Delay 模块。而对于快速率到慢速率的转换，则可以插入一个 Zero-Order Hold 模块。

(2) SingleTasking 模式。这个模式不检查模块间的速率转换，它在建立单任务系统模型时非常有用，在这种系统就不存在任务同步问题。

(3) Auto 模式。这种模式下，Simulink 会根据模型中模块的采样速率是否一致，自动决定切换到 multitasking 和 singletasking。如果模型中所有模块工作在相同的采样速率，那么就使用 singletasking 模式，反之工作在不同的采样速率，则使用 multitasking 模式。

## 7. 输出选项

solver 页的输出选项域允许用户控制 Simulink 产生的输出。它包括三个选项：Refine output（精细输出），Produce additional output（产生额外输出）和 Produce specified output only（只产生指定的输出）。

(1) Refine output 选项。这个选项可以理解成精细输出，其意义是在仿真输出太稀疏时，Simulink 会产生额外的精细输出，这一点就像插值处理一样。用户可以在 refine factor 设置仿真时间步间插入的输出点数。例如设置为 2，则在仿真采样时间点间产生 2 个额外输出。

产生更光滑的输出曲线，改变精细因子比减小仿真步长更有效。当精细因子变化时，Simulink 会根据一个扩展公式来计算在这些点的输出，但不会改变解法器仿真时采用的步长。精细输出只能在采用变长解法器时才能使用，并且在 ode45 效果最好。ode45 可以采用大的仿真步长。当用图形显示仿真输出时，就会发现曲线有时会很光滑，可以通过增大精细系数来解决这个问题，一个典型值是 4。

(2) Produce additional output 选项。它允许用户直接指定产生输出的时间点。一旦选择了这个项，那么在它的右边会出现一个 output times 编辑框，在这里用户指定额外的仿真输出点，它既可以是一个时间向量，也可以是表达式。与精细因子相比，这个选项会改变仿真的步长。

(3) 最后一个选项是 Produce specified output only。它的意思是让 Simulink 只在指定的时间点上产生输出。为此解法器要调整仿真步长以使之和指定的时间点重合。这个选项在比较不同的仿真时可以确保它们在相同的时间输出。

下面举一个例子来说明这三个选项的不同。

假定一个仿真在以下时间点产生输出：

{0 2.5 5 8.5 10}

当选择 refine output，并且它的精细因子为 2 时，产生输出的时间点为

{0, 1.25, 2.5, 3.75, 5, 6.75, 8.5, 9.25, 10}

如果选择 Produce additional output 项，并且说明指定的时间点为

```
>>[0:10]
```

则除了在 0、1、2、3、4、5、6、7、8、9、10 外还要在 2.5 8.5 产生输出，但如果选择 only，则只有 0 到 10。

6.2.2 Workspace I/O 页

这个页面的作用是定义将仿真结果输出到工作空间，以及从工作空间得到输入和初始状态。图 6-4 是它的全貌。

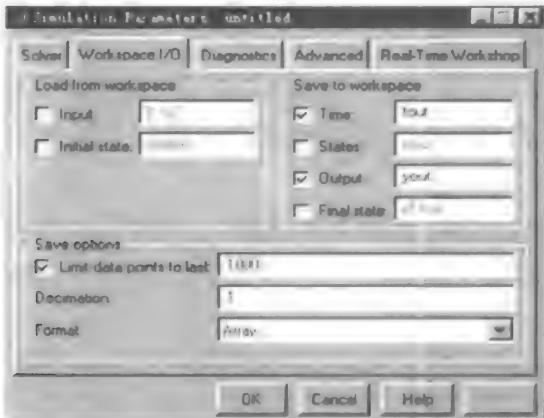


图 6-4 Workspace I/O 页

Workspace I/O 页被分成了三个面板：load from workspace、save to workspace 和 save options。它们各自的分工是：load from workspace 负责从工作空间获得输入和初始状态，save to workspace 负责保存变量到 MATLAB 工作空间，而 save options 则是让用户保存到工作空间的变量的格式。Workspace I/O 页主要以检查框为主要的控制风格，用户通过是否选中检查框来设置 Simulink 是否执行该选项所对应的功能。例如选中了 load from workspace 的 input 项，就表示要从 MATLAB 工作空间获得输入，而后面的编辑框则是包含输入值的变量名，只有在 input 被选中时，用户才能编辑变量名称。save to workspace 面板也是一样，用户只有通过检查框来确定是否输出这一项，然后才能自定义包含它的变量名。

1. 从 MATLAB 工作空间引入输入

单从参数的设置上而言，从 MATLAB 工作空间获得输入并没有什么特别的地方，但在运用中一定要注意输入变量的格式。

可以使用的外部输入格式有以下几种：外部输入矩阵、带时间的结构、结构和与端口对应的结构。

下面我们举一个从工作空间输入的例子来一一说明这些结构。

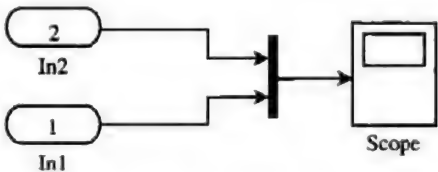


图 6-5 从工作空间输入示例

图 6-5 是一个简单的模型，它没有信号源，只有两个输入端口，in1 的输入宽度是 1，而 in2 的宽度是 2。这个模型的功能是从 MATLAB 工作空间获得输入，并显示在 scope 里。

首先来看看外部输入矩阵格式。请按照图 6-6 设置仿真参数的 workspace I/O 页，其实只需选中 input 就可以了，因为这时的缺省格式就是[t,u]。



图 6-6 参数设置

图中的形式定义了一个矩阵，时间是第一列，而输入 u 表示剩下的列，这就是所谓的外部输入矩阵格式。输入矩阵的第一列必须是升序的时间向量，这和仿真的时间演变顺序是一致的。而矩阵的其余列则是输入向量，每一列对应一个端口的输入，排列的顺序根据端口序数来确定，例如第一列就对应第一个输入端口。如果某个端口的信号宽度是大于 1，那它就对应着相应数目的列。于是矩阵的每一行就包含了某一次仿真时间点，以及当时的输入值。接着，就要在 MATLAB 工作空间定义变量 t 和 u 了。例如

```
>> t= (0:0.1:10)';    % 定义时间变量 t，注意' 的作用
                        % 是转置，把行向量变为列向量
>> u=[cos (t), sin (t), 4*cos (t)]; % 定义输入变量，因为输入
                        % 端的总宽度是 3，所以要定义 3 列。
```

定义完这些变量后，就可以运行仿真了，读者会发现在 scope 模块显示了这三条曲线。在定义外部变量时，要注意以下几点：

(1) 外部输入矩阵的列数是所有需从工作空间获得输入的端口的信号数（就是它们的宽度和）再加上 1。

(2) 存在多个端口时，输入信号与端口的对应是依赖于端口序数的。例如上面模型中，端口 1，in1 获得的输入是 cos (t)，而 in2 获得的输入是[sin (t), 4\*cos (t)]，它的宽度是 2。

Simulink 还可以从工作空间的 struct 格式的变量读取数据，为此要先在 input 的临近的编辑框设置结构名，如图 6-7 所示，“a”是本例中为外部输入结构取的名字，读者当然可以取别的名称。



图 6-7 结构变量名的设置

下面的问题就是如何在 MATLAB 工作空间定义结构 a。

```
>> a.time= (0:0.1:10)'; % 定义 struct a 的 time 字段
>> a.signals (1).values=[cos (t)]; % t= (0:0.1:10)，在前面示例中已经定义，故省略这一步
>> a.signals (2).values=[sin (t), 4*cos (t)];
```

这里，读者请记住 MATLAB 里定义变量不需要事先进行类型定义，a.time 就表示了新建立的变量是一个结构，time 是其中的一个字段，而不需要先定义 a 是一个结构类型的变量，它包括一个字段 time（在许多高级语言如 C 等都是需要预先进行定义声明）。关于结构的详



细用法请参阅 MATLAB 的帮助文档。

作为模型外部输入变量的结构必须具备特定的格式，Simulink 是这样规定的：

它必须具有两个字段：`time` 和 `signals`。`time` 字段保存代表仿真时间的向量，这就是“具有时间的结构”这一名称的来由。而 `signals` 的结构比较复杂，它是一个以结构作为元素的数组，数组的每一个元素对应一个输入端口。而每一个元素都是一个包含 `values` 字段的子结构，`values` 字段包含相应端口的输入值。如示例模型中，有两个字段，所以 `signals` 数组的大小是  $1 \times 2$ ，而对于端口 2，因为它的宽度是 2，所以它对应的输入值是两列的矩阵。读者可以在 MATLAB 子空间查询 `a`，看看它的组成。

```
>> a      %查询结构 a
a =
    time: [101x1 double]
   signals: [1x2 struct]

>> a.signals      %查询 signals 字段
1x2 struct array with fields:
    values

>> a.signals(1).values; % 这个命令就查询端口 1 的输入值
```

✎ 在定义作为外部输入的结构变量时，一定要注意各字段的拼法，它们是 `time`、`signals` 和 `values`，有时候可能会写错大小写，或者是少了 `s`，这都会导致运行出错。因为 MATLAB 是解释性的语言，它只会按照用户的意愿添加相应的字段，而不会提示出错。然而在 Simulink 从结构变量取相应的字段时，找不到该字段，就会出现错误。而且，Simulink 也不会关心该变量是否有它不需要的字段，于是可以不用重新清楚结构变量，就可以直接添加正确的字段进去。当然最好的方法是用 `clear` 清除掉原来的变量，再重新定义。

第三种可以被 Simulink 识别的外部变量形式是结构 `struct`，它与具有时间的结构的区别就在于前者的 `time` 字段是空的，即

```
>> a.time=[]      % 假定外部输入变量还是 a
```

至于 `signals` 字段的定义和后者一模一样。

也就是说模型不需要从外部获得仿真时间的输入，而是由在模型内的采样时间定义。但输入却是从 MATLAB 工作空间获得，获取的顺序是，在第一个仿真时间步就取该端口对应输入向量的第一个元素，第二个仿真步取第二个元素，依此类推。

还是以上面的模型作为例子，显然，在仿真参数对话框的设置和带时间的结构形式一样，可以不用去改变。只要改变 `a` 在 MATLAB 子空间的定义就可以了，为此请先清除变量 `a` 以前的定义（当然也可以在仿真参数对话框改用别的变量名称，从而不需清除）。

```
>> clear a;
>> a.time=[];
```

```
>> a.signals(1).values=[cos(t)];
>> a.signals(2).values=[sin(t),4*cos(t)];
```

而为了运行模型还必须对模型两个输入端口的参数进行改动。请双击输入端口，把它们 sample time 都改成 0.1，并去掉对话框中对 intoplate 检查框的选中。这样设置后运行模型，就不会有错误了。这里一定要把输入数据和仿真时间步的对应关系弄清楚。例如，分别把端口 1 和端口 2 的采样时间设为 0.2 和 0.05，得到的运行结果如图 6-8 所示。

图中出现的现象，很容易理解。因为在 MATLAB 工作空间的输入变量设定的输入值只有 100 个采样值，而在输入到模型时，Simulink 按照自己的采样时间步长依次把采样值输入。于是，对于输入端口 1，它的采样时间是 0.2，于是 100 个采样值就可以持续 20 秒，所以显示在 scope 的图形实际是  $\cos(t/2)$ ，而不是  $\cos(t)$ 。而对于输入端口 2，它的采样时间是 0.05，输入的 100 个采样点只能持续 5 秒的时间，对于剩下的仿真时间，Simulink 只能把它当成 0 来处理，显示波形相应的频率变为了原来的 2 倍。

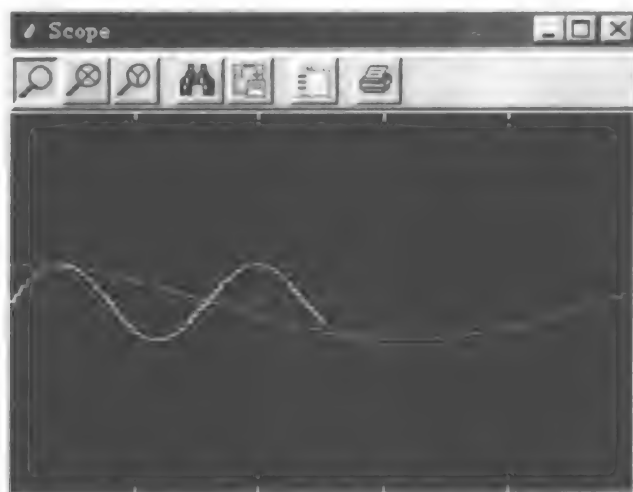


图 6-8 显示的结果

第四种被 Simulink 认可的外部数据输入格式是与端口对应的结构。这种格式，为模型的每一个端口设定一个外部输入变量，而每一个变量都是一个结构格式的变量，它们可以是带时间的，也可以是不带时间的。和前面的两种一样，这些结构都有一个 signals 字段，它里面就包含该端口的输入数据。在设置仿真参数时，需要把所有的结构变量按照所对应的端口的序数排列写在编辑框里。图 6-9 说明了如何对前面的示例模型定义这种格式。



图 6-9 参数设置

这就定义了对应输入端口 1 的外部输入变量是 d1，端口 2 对应的外部输入变量是 d2。下面分别把 d1 和 d2 设置为带时间和不带时间的结构。

```
>> t=(0:0.1:10)';
>> d1.time=t;
```

```
>> d1.signals.values=cos (t) ;  
>> d2.time=[];  
>> d2.signals.values=[sin (t) ,cos (4*t) ];
```

为此, 请读者设置好各端口的采样时间, 一般而言, 端口 1 的外部输入变量带时间, 所以可以把它的采样时间参数设为-1, 当然读者给它指定也可以, 这时 Simulink 就以用户指定的为准。而对于外部输入变量不带时间的端口 2, 则必须设定一个采样时间, 而且不能把它的值设为 0, 因为那是连续系统的采样时间。

## 2. 保存输出到工作空间

至于保存输出到 MATLAB 工作空间, 它要涉及到两个面板。Simulink 允许用户在 save to workspace 选择要返回到 MATLAB 工作空间的信号以及保存该量的变量名。它的设置形式和设置与外部输入基本上一样, 首先是选中检查框, 然后再指定返回变量名称。在面板上可以选择的有: time、states、output 和 final state。其中, time 和 output 是缺省被选中的, 因此一般运行一个仿真模型后, 在 MATLAB 工作空间都会增加两个变量 tout 和 yout, 而至于 states 和 final state 的概念读者在学习了 S-Function 的编写之后就会很清楚了。而 save option 的一些参数, 是用来说明返回变量的格式和输出数据的数量。其中的 limit rows to last 检查框的作用就是设定是否限制数据变量的行数, 选中时由后面的编辑框来设置具体的数目。Decimation 参数指定了一个亚采样因子, 它的缺省值是 1, 也就是对每一个仿真时间点产生值都保存, 而对于 2, 则是每隔一个保存一个。至于 format 参数, 它则是用来说明返回数据的格式。可以选择的格式有: matrix、structure 和 structure with time。它们的定义和前面的外部输入数据相应格式的定义是相同的。这里就不再赘述。

## 3. Loading and Saving States

可以保存和载入的状态, 有几种类型: 初始状态, 缺字状态和最终状态, 所谓的初始状态是指在仿真开始时各模块的状态, 相应的最终状态则指仿真结束时的状态。通过设置这里的参数可以从 MATLAB 工作空间导入初始状态。而通过把最终状态保存在变量里, 就可以使得这些状态能应用于其他的仿真。这个特性在用户想保存一个稳定状态并且在已知的状态重新开始仿真时, 非常有用。同输入或者输出一样, Simulink 能够识别的状态变量格式 (导入时), 对能生成的格式 (保存) 也有特殊的要求。在导入初始状态时, 相应的变量就要符合 Simulink 的要求, 而在保存状态时, 用户可以在 save option 面板的 format 参数来制定返回的格式。在名称上, 状态变量的格式和前面所讲是很类似的, 也是 matrix、struct with time 和 struct。除了 Matrix 格式差不多外, 应用于状态时, 另外两种格式还是有它们特殊的要求。

Struct with time 格式, 是带时间的结构格式, 它有两个最顶层的字段: time 和 signals。其中 time 的定义和前面的一样, 都是仿真时间向量。这里的 signals 字段也是一个子结构的数组, 数组里的每一个子结构都对应着模型中存在状态的一个模块。但这里的子结构要比前面所讲的复杂得多, 这种复杂性的增加是必须的, 也是可以理解的。因为 Simulink 在读取这些子结构时, 要判断它对应于哪个模块, 这和输入端口不同, 你无法事先对模块编号, 因此要开辟一个专门的字段来标识模块名。事实上, 每个子结构都有三个字段: values、label 和 blockName。value 字段包含该模块所有的状态向量值, 而 label 可以取值为 Cstate 或者 DState\_n, n 取值从 1 一直到相应模块具有的离散状态集的最大数目, 而 blockname 字段标识

了该子结构所对应的模块名，这个名称在模型图表就是模块的标签。

而 struct 格式，则是 struct with time 格式 time 字段为空时的特殊情况。

在 initial states 检查框没有被选中时，Simulink 使用模块参数里 initial conditions 参数设置取值。

当模型是多状态模型时，用户要设置初始状态，就有一个确定各状态的次序的问题。下面的命令可以做到这一点。

```
>> [sizes, x0, xstord] = sys ([], [], [], 0);
```

其中，sys 代表模型的名称，也就是保存模型的模型文件的名称。三个返回变量的意义如下：

sizes，反映了模型的某些特性的一个向量，只有前两个元素和初始状态有关，sizes (1) 代表连续状态的个数，sizes (2) 代表离散状态的个数。

x0，模块的初始条件。

xstord，一个包含模型中所有具有状态的模块的完全路径名称的字符串矩阵，xstord 中模块的顺序和 x0 相同。

例如，要获得 Simulink 提供的演示模型 vdp 的初始状态，可以使用如下的命令。

```
>> [sizes, x0, xstord] = vdp ([], [], [], 0)
```

```
sizes =
```

```
2
```

```
0
```

```
2
```

```
0
```

```
0
```

```
0
```

```
1
```

```
x0 =
```

```
2
```

```
0
```

```
xstord =
```

```
'vdp/x1'
```

```
'vdp/x2'
```

返回的这些变量所透露的信息为，vdp 模型有两个连续的初始状态，没有离散初始状态，模型中有状态的模块为 x1 和 x2 模块，它们的初始条件分别为 2 和 0。

6.2.3 Diagnostics 页

图 6-10 是 Diagnostics 页的全貌，读者可以按自己的兴趣去摸索上面的一些设置。Diagnostics 页分成两个部分：仿真选项和配置选项。配置选项下的列表框主要列举了一些常见的事件类型，以及当 Simulink 检查到了这些事件时给予的处理：什么也不做——none，还是发出警告信息——warning，或者给出错误提示——error。对于这些事件的处理方式，用户可以按自己的意愿设置。方法很简单，先用鼠标左键单击要设置的事件类型，然后在右边的单选按钮里选择相应的处理方式。

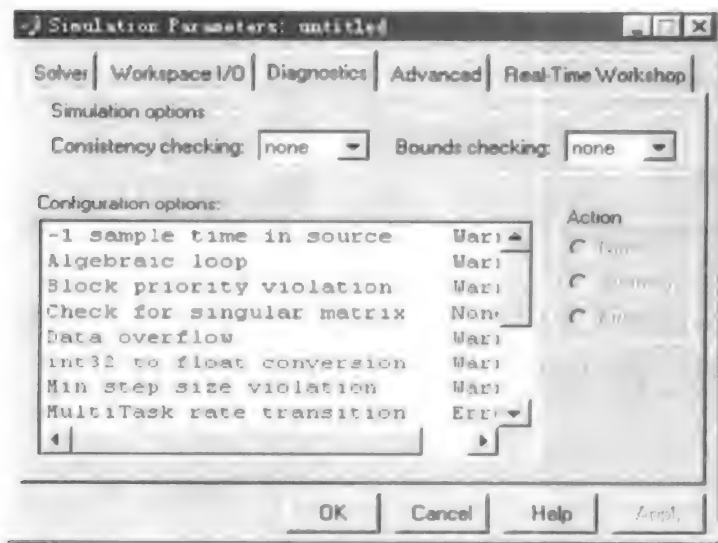


图 6-10 Diagnostics Page 诊断页

表 6-1 列出了列表框中列举的一些不正常事件的意义。

表 6-1 不正常事件的意义

事 件	说 明
-1 sample time in source	源模块设定采样时间为-1
Algebraic loop	Simulink 运行仿真时，检测到代数环
Check for singular matrix	Product 模块在矩阵乘法模式转化其中的一个输入时，检测到奇异矩阵
Data overflow	信号或者参数的值太大，以致它们的数据类型无法表示
int32 to float conversion	32 位整数被转化为浮点数，这种转换会产生精度的损失
Min step size violation	下一个仿真步小于模型所设定的最小仿真步，这可能发生在设定的误差限要求比最小步长要小的仿真步
Multitask rate transition	在运行在多速率模式下的两个模块间进行无效的速率转换
S-function upgrades needed	一个没有使用当前版本 S-函数特性的模块被遇到
Signal label mismatch	仿真遇到几个具有相同的源信号但不同的标签名的虚拟信号
Single Task ratet ransition	速率转换发生在运行在但速率模式下的两个模块间
Unconnected block input	模型包含具有没有连接的输入端的模块
Unconnected block output	模型包含具有未连接输出端的模块

续表

事 件	说 明
Unconnected line	模型包含未连接的直线
Unneeded type conversions	一个类型转换模块出现在类型转换不是必需的地方
Vector/Matrix conversion	向量到矩阵的转换或者矩阵到向量的转换发生在模块的输入端
Block Priority Violation	当运行模型时, Simulink 发现模块优先级设置错误

而 Options 面板里的几个检查框的意义是:

#### 1. Consistency Checking 选项

一致性检验是一种验证 Simulink ODE 解法器所作的几条假设的调试工具。它的主要作用是使 S-function 和 Simulink 以同种方式工作, 但是这种检查会导致仿真效率的急剧下降(最多可达 40%), 因此一般来说它总是被设为 off 的。但在使用 s-function 时, 一致性检验可以帮助用户判定产生错误的原因。一致性的另外一个目的, 是使模块在某个时间, 被调用时获得常数的输出, 这在解决 stiff 问题非常有用。

#### 2. Disabling Zero Crossing Detection 选项

选中这个选项禁止仿真时进行过零检测, 对于一个具有过零点的模型, 这样可以加快仿真的速度, 但会影响仿真结果的精度。一般它只用于那些本身具有内部的过零检测的模块, 它不能禁止 hit crossing 模块的过零检测。

#### 3. Disable optimized I/O storage

选中这个选项使 Simulink 为模块的每一个 I/O 值分配一个独立的缓存, 而不是复用同一个缓存。这样在仿真大的模型时, 会大大增加所需的存储空间。所以最好是在调试时才选中这个选项。特别是在以下几种情况, 读者要禁止缓存的复用:

(1) 调试一个 C-MEX 格式的 S-函数;

(2) 使用一个浮动 Scope 或者 Display 来监视所调试模型的信号。如果在使用一个缓存已被复用的浮动 Scope 或者 Display 时, 缓存复用依然可以使用, 那么 Simulink 就打开一个错误对话框。

#### 4. Relax boolean type checking 检查框

它的作用主要是为了和以前的 Simulink 版本相兼容。这样对于一个只能输入布尔数据类型的模块能输入 double 类型的数据。这一点很好理解。请看下面的示例模型(图 6-11)。

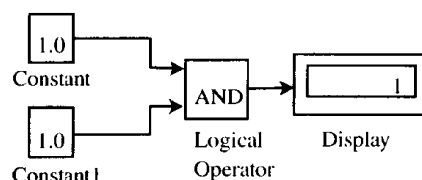


图 6-11 布尔类型检查示例模型

读者可以改变这个检查框的选中与否, 来试试运行仿真后的结果。读者可以发现在选中检查框时, 模型可以没有错误的运行, 但一去掉就会出现错误。

上面所述的 4 个选项在 Simulink3.0 都是放在诊断页的, 但在 Simulink4.0, 有一部分则

放在了 Advanced 页。

#### 6.2.4 Advanced 页

Advanced 页是 Simulink4.0 新增加的页，它上面列置了原来放在诊断页的一些选项设置。图 6-12 显示了它的样子。

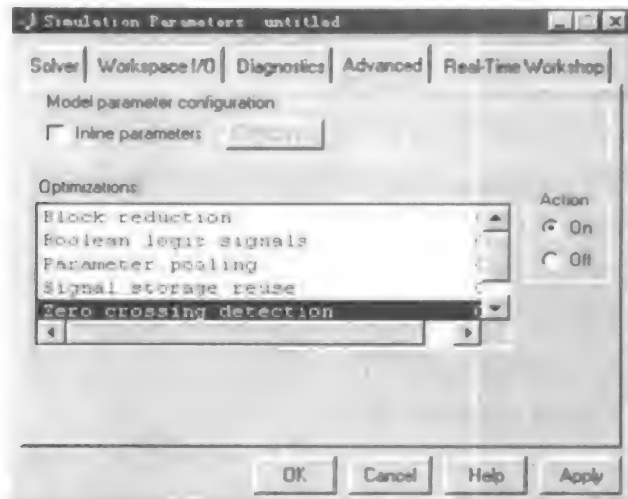


图 6-12 Advanced 页

这个页分为两个面板：一个是模型参数配置面板，一个则是优化面板。

在模型参数面板，读者可以选中 Inline parameters 检查框。这样将使所有的参数变成不可调谐的（在仿真运行时变化），除了那些用户特别设定的以外。将参数设为不可调谐，会使 Simulink 把它们当成常数，这样就可以加速仿真的运行。使用 Configuration dialog box 对话框可以设定在 Inline parameters（内嵌参数）选中时，用户想保留为可调谐的变量。这个对话框可以通过检查框旁边的 configure 按钮来打开。图 6-13 是它的示意图。

这个对话框的使用是十分简单的，读者只需在对话框左边源列表里的变量中，选择需设定为可调谐的，用左下角的 Add to table 按钮，就可以添加到右边的全局列表里，也就变成了可调谐的（全局的）。至于列表中的 Storage Class 和 Storage type qualifier 两个属性，都是用于代码生成的，读者可以在本书的第十章找到它们的说明。

当这个选项被选中时，只有符合下列条件的参数才能在仿真过程中变化：

- (1) 参数的值必须是在 MATLAB 工作空间的变量；
- (2) 模型参数必须是在模型参数配置对话框设定为 global 的变量。

改变符合上述条件的参数，可以通过改变相应的工作空间的变量值，并且用 Edit 菜单下的 Update Diagram 命令来更新图表。

而在优化面板，有一些选项在 Simulink3.0 中是位于诊断页的，如 Boolean logic signals, Signal storage reuse 和 Zero-crossing detection, 关于它们的解释请看前面一小节。除此之外的两个选项的意义如下：

- (1) Block reduction. 用一个合成的模块来替代一组模块，提高仿真速度；
- (2) Parameter pooling. 这个选项用于代码生成，在不进行代码生成时，这个选项可以不去考虑。

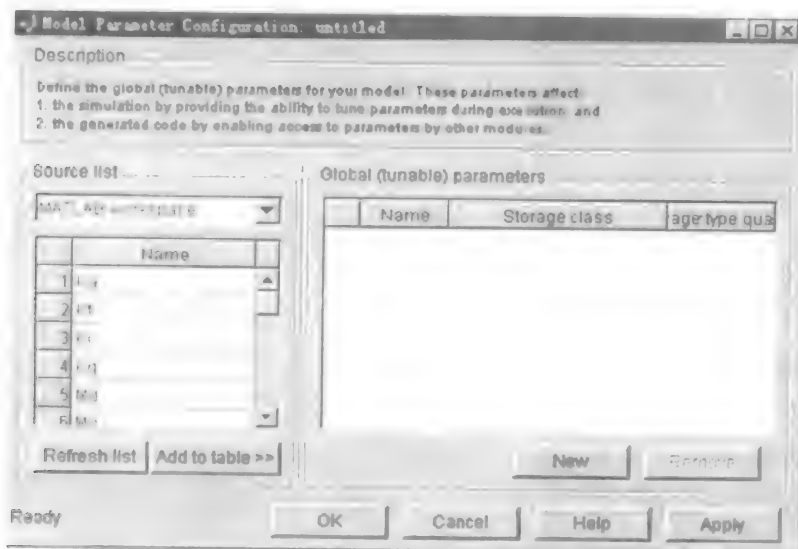


图 6-13 参数配置对话框

## 6.3 改善仿真的性能和精确度

仿真的性能和精确度受很多的事情的影响，包括模型的设计和仿真参数的选择。对于大多数问题，使用缺省的仿真参数值、解法器可以精确而有效的解决。但有些模型，适当的调解法器和仿真参数，它们可以得到更好的仿真结果。并且，如果用户知道模型行为的信息，并把它告诉解法器，也会提高性能。

### 6.3.1 加速仿真

一个模型的仿真速度过慢，是由许多因素造成。下面是其中的一些：

(1) 模型包括一个 MATLAB Fcn 模块。当执行一个包含 MATLAB Fcn 模块的模型，Simulink 在每一个仿真时间步都要调用 MATLAB 解释器。所以应该尽可能的使用 Simulink 的内置 Fcn 模块或者是最基本的 math 模块。

(2) 模型包含用 M 文件的 S-函数，M 文件 S 函数同样会使 Simulink 在每一个仿真时间步调用 MATLAB 解释器。替代的方法是把 M 文件 S-函数转化成 c-mex 函数或者是建立一个等价的子系统。

(3) 模型包含一个存储模块。使用存储模块将使阶数可变的解法器(如 ode15s 和 ode113)在每个仿真时间步被重置回 1 阶。

(4) 仿真的时间步长太小。解决的方法是把最大仿真步长参数设置回 Simulink 的缺省值——auto，看看效果如何。

(5) 仿真的精度要求过高。一般来说，相对误差限设为 0.1%就已经足够。当模型存在取值会趋向于零的状态，仿真时如果绝对误差限度太小，会使仿真在在接近零的状态附近耗费过多的仿真步。

(6) 仿真的时间过长。可酌情减小仿真的时间间隔。



(7) 所解决的问题是 stiff 问题，却选择了一个非 stiff 的解法器。可以试试 ode15s。

(8) 模型所设置的采样时间的公约数过小，这样使 Simulink 可采用的基准采样时间过小，因为 Simulink 会选择足够小的时间步确保所设置的采样点都能取到。

(9) 模型包含一个代数环。代数环的求解方法就是在每一个时间步迭代地进行计算，这当然会严重地降低仿真的性能。

(10) 模型把一个 random number 模块作为 integrator 模块的输入。对于连续系统，可以使用 source 子库里的 Band-Limited WhiteNoise（带限白噪声）模块。

### 6.3.2 改善仿真的精度

检验仿真精度的方法是，修改仿真的相对误差限和绝对误差限，在一个合适的时间跨度反复运行仿真，看看仿真的结果有没有大的变化，如果变化不多，则表示解是收敛的。

如果仿真在开始时错过了模型的关键行为，那么可以更改初始步长使仿真不会忽略这些关键的行为。

如果仿真的结果不稳定，可能是以下原因。

(1) 系统本身不稳定；

(2) 如果正在使用 ode15s，用户可以把最大阶数定为 2 或者尝试 ode23s；

(3) 如果仿真的结果看起来不是很精确，它的原因可能是：

- 模型有取值接近零的状态，如果模型的绝对误差限过大，会使仿真在接近零的区域运行的仿真时间步太少。解决的办法是修改绝对误差参数或者在积分模块的对话框修改初始的状态；

- 如果改变绝对误差限还不能达到预期的误差限，请修改相对误差限，使可接受的误差降低，并减小仿真的步长。

## 6.4 从命令行运行仿真

相对于使用菜单命令，从命令行运行仿真使得用户可以从 M 文件来运行仿真，这样就允许仿真和模块参数可以被不断地改变参数。也就可以让用户随机地改变参数和循环地运行仿真，就可以进行蒙特卡罗分析了。有两个命令可供用户选用：sim 和 set\_param 命令。

### 6.4.1 使用 sim 命令

sim 命令的作用是仿真一个由 Simulink 模型表示的系统，它的完整格式是

```
[t,x,y] = sim (model, timespan, options, ut) ;
```

或者

```
[t,x,y1, y2, ..., yn] = sim (model,timespan,options,ut) ;
```

其中，只有 model 参数是要求的，右边其他的参量，都被允许置为空矩阵（[]）。命令中设置过的参数将会覆盖在模型中定义的参数值（如用仿真参数对话框等等）。当仿真的模型是连续系统，那么命令中还必须设定 solver 参数，这可以使用 simset 命令。

下面是各个参量的详细说明。

**t** 返回仿真的时间向量。  
**x** 返回仿真的状态矩阵，排列顺序是先连续状态，然后是离散状态。  
**y** 返回仿真的输出矩阵，其中的每一列对应着一个根层次的输出端口（即顶层系统），顺序按端口数字对应。如果一个输出端口的结果是向量信号，那它相应的占有合适的列数。

**y1,...,yn**

返回模型中的根层次输出端口的输出，这样的端口模型有 **n** 个。

**model** 一个模块图表的名称。

**timespan** 仿真的起始和终止时间。有两种说明方式：**tFinal** 仅指定结束时间，而起始时间为 0；**[tStart tFinal]** 说明开始和结束时间。

**options** 由 **simset** 命令建立的结构，用于指定可选的仿真参数。

**ut** 可选的对顶层输入端口模块的外部输入。**ut** 可以是一个 **MATLAB** 函数（用 **string** 表达），指定每一个仿真时间步的输入 **u=UT(t)**，**UT** 表示输入和 **t** 的关系，或者是一个用逗号分割的列表，**ut1, ut2, ……**，其中的每一个对应着一个输入端口。面向所有端口表格化输入可以是 **MATLAB** 矩阵或者结构的形式。面向单个端口的表格化输入只能是结构的形式。

下面是它的一个使用示例。它仿真 **Van der Pol** 方程模型，这个命令全部使用缺省参数：

```
>> [t,x,y] = sim('vdp') % vdp 是 Simulink 的演示模型
```

而下面的命令，使用了 **vdp** 模型图标上的参数，但是定义 **Refine** 参数的值：

```
>> [t,x,y] = sim('vdp', [], simset('Refine',2));
```

第三个命令仿真 **vdp** 模型 1000 秒，并保存返回变量的最后 100 行，仿真的输出变量仅包含 **t** 和 **y**，但在变量 **xFinal** 保存最终状态向量。

```
>> [t,x,y] = sim('vdp', 1000, simset('MaxRows', 100, 'OutputVariables', 'ty', 'FinalStateName', 'xFinal'));
```

#### 6.4.2 使用 **set\_param** 命令

可以使用 **set\_param** 命令来开始，结束，暂停或者继续仿真，或者更新模块图表。类似的，还可以使用 **get\_param** 命令来检查一个仿真的状态。**set\_param** 命令的使用格式是

```
set_param('sys', 'SimulationCommand', 'cmd')
```

其中，**'sys'** 是系统的名称，**'cmd'** 是控制命令取值有：**'start'**，**'stop'**，**pause**，**'continue'**，或者 **'update'**。

**get\_param** 命令的使用格式是：

```
get_param('sys', 'SimulationStatus')
```

这个命令的返回值可以为：**'stopped'**，**'initializing'**，**'running'**，**'paused'**，**'terminating'** 和 **'external'**（见本书的第十章）。

## 6.5 分析仿真结果

### 6.5.1 观看输出结果的轨迹

Simulink 中，观看仿真的输出轨迹，有三种方法：

- (1) 把信号输入到 scope 模块或者 xy graph 模块；
- (2) 把输出写入返回变量，并用 MATLAB 命令绘图；
- (3) 使用 To Workspace 把输出写入到工作空间，然后用 MATLAB 命令绘制出仿真的图。

#### 1. 使用图形显示模块

第一种方法的 scope 模块在前面，已经多次使用，这里就不多讲了，这里只谈一下 xygraph 模块和 scope 的区别。在 scope 模块，作出的图是以时间为横坐标，输出作为纵轴。而 xygraph 则可以绘制出一个信号相对于另一信号的图形。

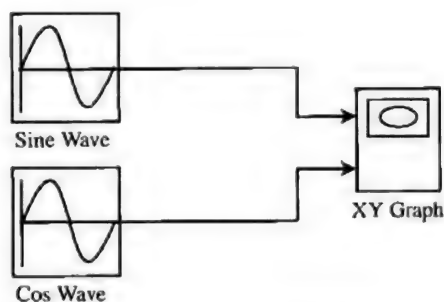


图 6-14 XY Graph 模块使用

图 6-14 中的 cos wave 模块是将 sinewave 的相位设为  $\pi/2$  得到，这样 xy graph 模块显示的图形为图 6-15。

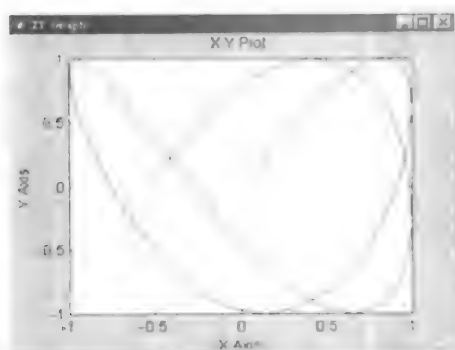


图 6-15 XY Graph 模块显示的图形

上述图形就是物理上讲的李萨如图形，在判断信号是否同步时有很大的用处。

#### 2. 使用返回变量

使用返回变量的首要步骤是在设置仿真参数对话框里，选中输出检查框 time 和 output，再把相应的时间变量和保存输出的变量设置好，一般而言，它们缺省值被选中，而且变量名

分别为 tout 和 yout。下面来看一个例子，首先请按图 6-16 建好示例模型。

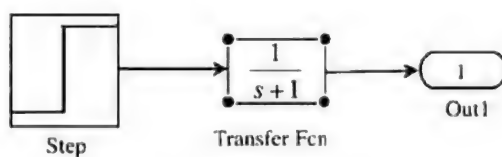


图 6-16 示例模型

其中 step 模块在 source 子库，transfer fcn 模块在 continuous 子库，模型中的输出端口是必须的，它表示它对应的信号就是上输出信号。运行仿真后，读者会发现 MATLAB 工作空间多了两个变量：tout 和 yout，这就是由 Simulink 保存在 MATLAB 工作空间的返回变量。用 plot 命令就可以绘出所需的图形。

```
>> plot(tout,yout);
```

### 3. 使用 to workspace 模块

最后一种方法是使用 to workspace 模块，它在 sinks 子库，还是用上面的例子。请读者把 out1 模块用 to workspace 模块替代（图 6-17）。

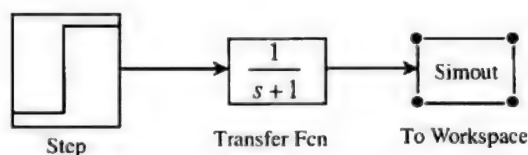


图 6-17 替换后的模型

to workspace 模块的参数对话框如图 6-18 所示。

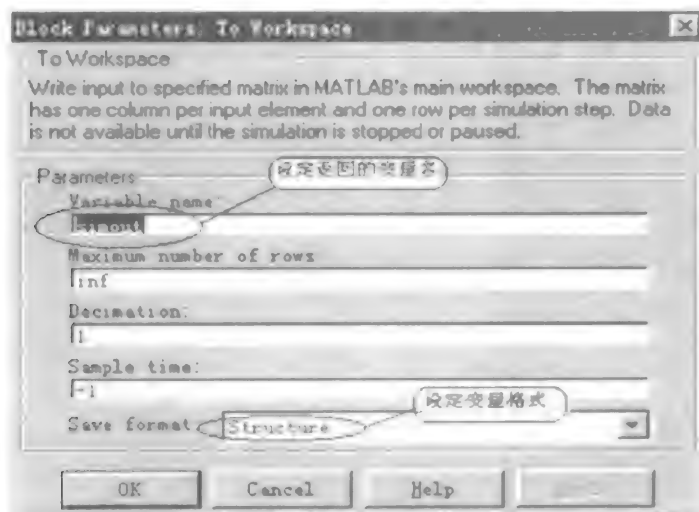


图 6-18 to workspace 的参数对话框

图上标出使用 to workspace 模块时两个比较重要的参数。请运行仿真

```
>> who
```

Your variables are:

```
simout    tout
```

```
>>simout
```

```
simout =
```

```
time: []
```

```
signals: [1x1 struct]
```

```
blockName: 'testreturnout/To Workspace'
```

```
>> plot (tout,simout.signals.values) ;
```

## 6.5.2 线性化

### 1. 线性化示例

Simulink 提供了 `linmod` 和 `dlinmod` 两个函数来抽取用状态空间矩阵形式  $A$ 、 $B$ 、 $C$  和  $D$  表示的线性模型。状态空间矩阵按下面的方程来描述线性的输入输出关系：

$$\dot{x}' = Ax + Bu$$

$$y = Cx + Du$$

其中， $x$ ， $u$ ， $y$  分别表示状态、输入和输出向量。例如，图 6-19 是称为 `lmod` 的模型，其中 `Plant` 模块是把 `Transfer Fcn` 模块的 `denominator` 参数设为 `[1 2 1]` 得到。

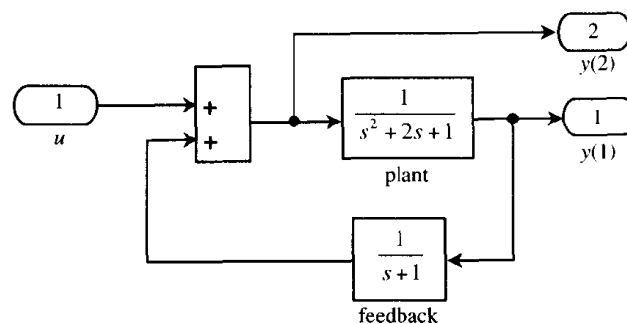


图 6-19 lmod 模型图表

保存模型后，用下面的函数就可以把模型变成线性状态方程的形式。

```
>> [A,B,C,D]=linmod ('lmod') % lmod 是模型的名称
```

```
A =
```

```
-2    -1     1
 1     0     0
 0     1    -1
```

```
B =
```

```
1
0
0
```

```
C =
```

```

0    1    0
0    0    1
D =
0
1

```

一旦数据具有状态空间的形式，或者被转化为 LTI 对象，用户可以使用控制系统工具箱里的函数进行进一步的分析：

- 将状态空间转换为 LTI 对象，使用

```
sys = ss (A,B,C,D)
```

- 绘制波特相位幅频图，使用

```
bode (A,B,C,D) 或者 bode (sys)
```

- 求出线性化时间响应，可以使用的命令有：

```
step (A,B,C,D) 或者 step (sys), 求单位阶跃响应；
```

```
impulse (A,B,C,D) 或者 impulse (sys), 求系统的单位冲击响应。
```

下面举一个例子来说明如何对一个系统进行线性化，并据此求出它的时间响应曲线。首先使用 `linmod` 对 f-14 进行线性化：

```

>> [A,B,C,D]=linmod ('f14')
>> t=0 :1:10;
>> y=step (A,B,C,D,1,t) ;
>> plot (t,y) ;

```

这样，就可以绘出通过对 f-14 的线性化系统对 f-14 进行线性化仿真，在上面的命令中，使用 `step` 求出了系统的阶跃输出响应。

如果模型本身是非线性的，那么要选取一个工作点，用来标识从哪里抽取线性化的模型。非线性模型同样对摄动的大小很敏感，所以必须存在一个截断误差和舍入误差的折中。`Linmod` 函数提供了额外的参数来说明工作点和扰动的大小。使用的命令为

```
[A,B,C,D] = linmod ('sys', x, u, pert, xpert, upert)
```

对于离散系统或者离散和连续的混合系统进行线性化，就要使用 `dlinmod` 函数，它的语法和 `linmod` 基本类似，只是使用时必须包括一个说明采样时间的参数。

```
[A,B,C,D]=dlinmod ('SYS',TS)
```

更详细的信息请看 `dlinmod` 的帮助信息。

用 `linmod` 函数对包含微分模块或者传送时延模块的模型进行线性化，将会出现问题。因此，在进行线性化之前，须先用经过特殊设计的能避免这个问题的模块替代上面的微分模块和传送时延模块。这些替代模块的位置是在 `SimulinkExtras` 的 `Linearization` 子库。它们是：

- (1) 对于 `Derivative` 模块，请用 `Switched derivative` 模块来替代。

(2) 对于 Transport Delay 模块, 请用 Switched transport delay 模块来替代 (这个模块只有在安装了控制系统工具箱, 才能被使用)。

对于微分模块的另外一种处理方式是, 尽量把微分项合并到其他的模块中。这句话的意思是, 例如, 对于一个微分模块后串接一些 Fcn 模块的实现形式, 用一个具有传递函数  $s/(s+a)$  的 Fcn 模块的形式来实现会更好一些 (尽管这并不总是可以办到的)。与如图 6-20 所示的一样, 右边的这种串接形式用左图单个 Fcn 模块实现会更好些。

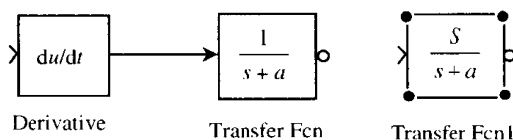


图 6-20 用传递函数来替换微分模块

## 2. 线性化函数使用方法

linfun 支持的调用格式有:

`[A,B,C,D] = linfun ('sys')`

`[A,B,C,D] = linfun ('sys', x, u)`

`[A,B,C,D] = linfun ('sys', x, u, pert)`

`[A,B,C,D] = linfun ('sys', x, u, pert, xpert, upert)`

其中各参量的意义分别为:

**linfun** 线性化函数, 包括: `linmod`, `dlinmod`, `linmod2`。

**sys** 需要线性化的系统名称。

**x 和 u** 状态和输入向量如果定义这些量, 那么这些量就设置了工作点, 在这个点上提取线性模型。

**pert** 可选标量, 用来设置 x 和 u 的干扰。如果这个值没有定义, 就采用默认值 `1e-5`。

**xpert 和 upert**

可选向量, 用来为每一个状态和输入设置扰动。如果定义了这个向量, 那么就忽视双面上面的标量 `pert`, 这种情况下, 状态 x 的第 i 个元素收到扰动后就变成 `x(i) + xpert(i)`; 输入 u 的第 j 个元素收到扰动后就变成 `u(j) + upert(j)`。

现对它的使用说明如下。

`linmod` 得到的是用常微分方程描述的 Simulink 模型的线性模型。返回的模型用状态空间 A、B、C、D 形式来表示其输入和输出的关系, 这样的线性模型可用下面的式子来表示:

$$\dot{x}' = Ax + Bu$$

$$y = Cx + Du$$

`[A,B,C,D] = linmod ('sys')` 可以得到在状态变量 `x=0` 和输入 `u=0` 这个工作点附近的线性化模型。`linmod` 是在工作点附近对状态施加扰动后来确定状态确定状态导数和输出的变化速率 (即雅克比矩阵), 并把所得的结果用来计算状态空间矩阵。每一个状态受到扰动后变成

$$x(i) + \Delta(i)$$

其中,

$$\Delta(i) = \delta(1 + |x(i)|)$$

同样, 第  $j$  个输入受到扰动后, 变成

$$u(j) + \Delta(j)$$

其中,

$$\Delta(j) = \delta(1 + |u(j)|)$$

### 3. 离散系统的线性化

`dlinmod` 能够以任意给定的采样时间对离散系统、多速率系统, 以及连续和离散这类混合系统进行线性化。除了要求在第二个选项传入插入采样时间来对系统线性化外, `dlinmod` 的调用格式和 `lmod` 基本相同, 例如:

```
[Ad,Bd,Cd,Dd] = dlinmod('sys', Ts, x, u);
```

上面这个命令可以产生一个以状态向量  $x$  和输入向量  $u$  为工作点、采样时间为  $T_s$  的离散状态空间模型。要想得到一个离散系统的近似连续模型, 只要把  $T_s$  设为 0 即可。如果一个模型是由线性模块、多速率模块、离散模块和连续模块组成的, 那么在同时满足下列条件的情况下, `dlinmod` 产生一个采样时间为  $T_s$ 、且具有相同频率和时间响应的线性模型:

- (1)  $T_s$  是系统中所有采样时间的整数倍;
- (2)  $T_s$  是不小于系统中最慢的采样时间;
- (3) 系统是稳定的。

不过, 在上面这些条件不满足的情况下, 也有可能得到有效的线性模型。要看一个系统是否稳定, 实际上只要计算线性化后所得矩阵  $Ad$  的特征值就可以了。因此, 如果  $T_s > 0$  而且特征值在单位圆里, 即

```
all(abs(eig(Ad))) < 1 % eig 表示求矩阵 Ad 的特征值
```

那么系统是稳定的。同样, 如果  $T_s = 0$ , 而且所有的特征值在左半平面, 那么系统也是稳定的。即

```
all(real(eig(Ad))) < 0
```

当系统不稳定且采样时间不是其他采样时间的整数倍时, `dlinmod` 就可能产生复矩阵  $Ad$  和  $Bd$ 。然而在这种情况下, 仍然可以通过矩阵  $Ad$  的特征值来验证系统的稳定性。

### 6.5.3 平衡点的分析

#### 1. 确定平衡点

所谓平衡点, 也称为 `trim` 点, 就是指参数空间中使动态系统处于稳定状态的点, 在数学上, 平衡点就是使状态的导数为 0 的工作点, 既包括输入值  $u$  和状态值  $x$ 。例如, 飞行中的飞机的平衡点就是使飞机竖直和水平飞行的控制设置。

确定系统的稳态平衡点的函数是 `trim`, 还是以上面的 `lmod` 模型 (图 6-21) 为例来演示如何确定平衡点。例如, 用 `trim` 来寻找使两个输出都为 1 的输入和状态值。



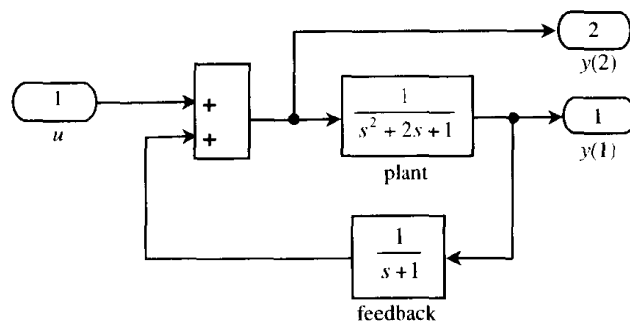


图 6-21 lmod 模型

首先，对状态和输入值进行一个初步的猜测，并把预期的输出值赋给  $y$ 。

```
>> x = [0; 0; 0];
>> u = 0;
>> y = [1; 1];
```

然后要用索引变量来规定模型的输入、输出和状态中哪些可以变化，哪些不能变化。

```
>> ix = [];           % 不固定状态变量中任何一个
>> iu = [];           % 不固定输入
>> iy = [1; 2];       % 固定输出端口 1 和输出端口 2
```

调用 `trim` 命令来求出稳定点

```
>> [x,u,y,dx] = trim('lmod',x,u,y,ix,iu,iy)
```

```
x =
    0.0000
    1.0000
    1.0000
u =
    3.0537e-016
y =
     1
     1
dx =
    1.0e-015 *
         0
    0.0402
    0.2220
```

## 2. trim 命令使用格式

下面就来详细介绍 `trim` 函数的用法，`trim` 函数支持的格式有：

```
[x,u,y,dx] = trim('sys')
```

```

[x,u,y,dx] = trim ('sys',x0,u0,y0)
[x,u,y,dx] = trim ('sys',x0,u0,y0,ix,iu,iy)
[x,u,y,dx] = trim ('sys',x0,u0,y0,ix,iu,iy,dx0,idx)
[x,u,y,dx] = trim ('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options)
[x,u,y,dx] = trim ('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options,t)
[x,u,y,dx,options] = trim ('sys',...)

```

我们知道 trim 命令就是获得使状态的微分为零的输入 u 和状态 x 的值。trim 命令的工作原理大致是，从一个初始值开始（调用时，需要先指定），然后使用二分查找的算法来搜索，直至找到最接近的 trim 点。当 trim 命令找不到平衡点，它就会返回在搜索中遇到的使状态微分在 min-max 意义下最接近零的点。即是使距微分的零点的最大偏离最小的点。trim 命令可以找出符合特定输入、输出状态条件的 trim 点，和让系统按指定方式变化的点。也就是，使系统的状态微分等于特定的非零值。

`[x,u,y] = trim ('sys')` 找到距系统初始状态 x0 最近的平衡点，特别的，trim 找出使  $[x-x_0, u, y]$  的最大绝对值最小的平衡点。

如果找不到这样的平衡，它将返回使 `abs(dx)` 最小的点。这时就可以使用下面的命令来获得 x0 的值。

```
>> [sizes, x0, xstr] = sys ([],[],[],0)
```

`[x, u, y, dx] = trim ('sys',x0,u0,y0)` 为状态 x、输入 u 和输出 y 定义了各自的猜测值。在这种情况下，trim 命令就使  $[x-x_0; u-u_0; y-y_0]$  的最大绝对值达到最小。

x, u, y 的每一个元素可用下面的调用格式进行固定：

```
>> [x, u, y, dx] = trim ('sys', x0, u0, y0, ix, iu, iy)
```

整数向量 ix、iu 和 iy 指出对 x0、u0 和 y0 的哪一个元素进行固定。既然无法保证平衡点一定存在，因此问题就转化为寻找一个稳态值使得

```
abs ([x(ix)-x0(ix); u(iu)-u0(iu); y(iy)-y0(iy)])
```

的最大绝对值达到最小。

trim 在限制状态倒数为零的情况下，用一个有约束的优化算法来求解一个由 x、u 和 y 的希望值所组成的最大绝对值最小问题。对于这样一个问题，有可能没有可行解。在这种情况下，trim 就在状态倒数偏离 0 这种最坏条件下，使上面的最大绝对值达到最小。

要使状态微分固定为一个非零值，可以采用

```
>> [x, u, y, dx] = trim ('sys', x0, u0, y0, ix, iu, iy, dx0, idx)
```

其中 dx0 表示希望的偏离值，idx 用来表示对 dx 中的哪一个元素进行固定。

### 3. 举例

考虑如下的一个线性空间模型

$$\dot{x}' = Ax + Bu$$

$$y = Cx + Du$$

假定这个模型的名字为“trim\_example”，读者可以先用 Simulink 建立这个模型，它的图表如图 6-22 所示，模型中主要的模块就是 state space 模块，它在 continuous 库里。

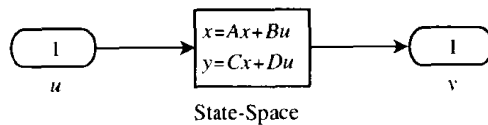


图 6-22 trim\_example 模型图表

其中，模块的各个参数分别为：

$$A = [-0.09 \ -0.01 \ 1 \ 0];$$

$$B = [0 \ -7 \ 0 \ -2];$$

$$C = [0 \ 2 \ 1 \ -5];$$

$$D = [-3 \ 0 \ 1 \ 0];$$

例 1 要寻找一个系统的平衡点，可用下面的命令：

```
>> [x,u,y,dx,options]=trim('trim_example')
```

```
x =
```

```
0
```

```
0
```

```
u =
```

```
0
```

```
0
```

```
y =
```

```
0
```

```
0
```

```
dx =
```

```
0
```

```
0
```

还可以求出所进行迭代的次数

```
>> options(10)
```

```
ans =
```

```
7
```

例 2 要寻找系统在  $x=[1; 1]$  和  $u=[1; 1]$  附近的一个平衡点，可用下面的命令：

```
>> x0=[1;1];
```

```
>> u0=[1;1];
>> [x, u, y, dx, options] = trim ('trim_example', x0, u0) ;
>> x,u,y,dx,options (10)
```

```
x =
    1.0e-013 *
    -0.2770
    -0.2771
```

```
u =
    0.3333
   -0.0000
```

```
y =
   -1.0000
    0.3333
```

```
dx =
    1.0e-013 *
    0.9783
   -0.0053
```

```
ans =
    31
```

例 3 要寻找一个系统输出固定为 1 的平衡点，可用下面的命令：

```
>> y = [1;1];
>> iy = [1;2];
>> [x, u, y, dx] = trim ('trim_example', [], [], y, [], [], iy) ;
>> x,u,y,dx
```

```
x =
    0.0009
   -0.3075
```

```
u =
   -0.5383
    0.0004
```

```
y =
    1.0000
    1.0000
```

```
dx =
```

```
1.0e-016 *  
0.0043  
0.4369
```

要寻找一个系统输出固定为 1、状态倒数分别为 0 和 1 的平衡点，可用下面的命令：

```
>> iy = [1;2];  
>> dx = [0;1];  
>> idx = [1;2];  
>> [x,u,y,dx,options] = trim ('trim_example',[],[],y,[],[],iy,dx,idx) ;  
>> x,u,y,dx  
x =  
0.9752  
-0.0827  
u =  
-0.3884  
-0.0124  
y =  
1.0000  
1.0000  
dx =  
-0.0000  
1.0000  
>> options (10)  
ans =  
13
```

## 第七章 Simulink 调试器

“人非圣贤，孰能无过”。无论是多么熟练、多么睿智的用户，也不可能做到在建立模型时一点差错不出。正像许多高级编程语言都提供了功能强大的调试器一样，Simulink 也提供了一个工具便于用户查找和诊断模型中的错误。它允许用户通过单步运行仿真和显示模块的即时状态、输入和输出，来查明错误。这一章就为读者介绍如何使用 Simulink 调试器来诊断 Simulink 中出现的问题。在进入具体的细节前，请读者把以前在使用高级编程语言如 vc 等接触到的调试器的印象淡化，Simulink 的调试器和它们相比就略显简陋，Simulink3.0 以前的调试器都基于命令行输入，而不是图形用户界面，所以在使用上有时可能不大方便。最新推出的 Simulink4.0 版本，提供了一个图形化的调试界面，简化了调试操作，但它在功能上和以前版本的调试器没有什么变化，只是为以前的一些命令提供了图形化的操作。本章在讲解的时候为了兼顾使用以前版本的读者，首先介绍各个仿真命令，这些命令适用于所有版本的 Simulink。最后介绍一下 Simulink4.0 的图形调试界面，而它的使用就比较简单了。

### 7.1 使用调试器

#### 1. 启动调试器

使用 `sldebug` 命令或者 `sim` 命令的 `debug` 选项在调试器控制下启动一个模型。例如可以在 MATLAB 命令窗口输入

```
>> sim('vdp',[0,10],simset('debug','on'));
```

或者

```
>> sldebug 'vdp' ;
```

这两个命令把 Simulink 的示例模型 `vdp` 载入内存并且在第一个仿真时间步暂停在执行顺序中的第一个模块。调试器加亮显示模型的起始模块和模型图表中与之相关联的输出信号线。图 7-1 显示了 `vdp` 在调试器模式启动时的模块图表。

调试器还在 MATLAB 命令窗口显示仿真的开始时间和调试命令提示符。命令提示符显示模块的索引和将要被执行的第一个模块的名称。例如前面输入的命令会在命令窗口产生下面的输出。

```
[Tm=0                                ] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/x1') :
```

这时，读者就可以通过在提示符后输入调试器命令或者其他的 MATLAB 命令进行操作，

如获得调试器的帮助、单步运行仿真和检查数据等等。下面就来介绍如何使用这些调试器命令。

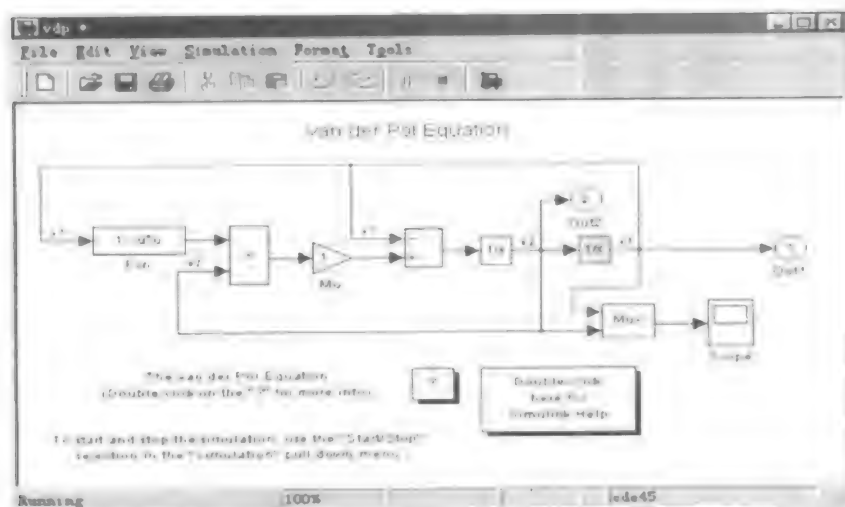


图 7-1 调试模式启动时的 vdp 模型

用户可以通过在提示符后输入 **help** 命令来获得调试器命令的简短描述, 而对于每一个命令的详尽描述可以查阅本章的调试器命令参考一节。

(sldebug @0:0 'vdp/x1'):help % 以后除非特别说明, 本章中的 >> 符号都表示在提示符的输入

### Commands:

step	Step to next block.
next	Go to start of next time step.
disp [s:b   gcb]	Register or display I/O of block (s) at every stopping point.
undisp <s:b   gcb>	Remove a display point.
trace <s:b   gcb>	Add a trace point to display block I/O as it is executed.
untrace <s:b   gcb>	Remove a trace point.
probe [s:b   gcb]	Probe I/O of block.
break <s:b   gcb>	Break before block is executed.
bafter <s:b   gcb>	Break after block is executed.
bshow s:b	Show block in system, s, with sorted list index, b.
clear <s:b   gcb>	Clear break point.
zcbreak	Toggle: break at nonsampled zero crossing events?
zclist	Display the nonsampled zero crossings list.
xbreak	Toggle: break when step size is limited by a state?
tbreak [t]	Set/clear time break point.
nanbreak	Toggle: break on non-finite (NaN, Inf) values.
continue	Continue the simulation.
run	Stop debugging and finish the simulation.

stop	Stop execution.
quit	Abort the simulation.
status [all]	Display debugging actions in effect.
states	Display current state values.
systems	Display a list of the model systems.
.....	(略去一部分, 笔者注)

调试器允许用户输入调试命令的缩写形式, 关于各命令对应的缩写请读者查看调试命令参考。Simulink 还允许用户用空命令来重复前面执行过的命令。例如, 读者输入 **step** 命令单步执行 vdp 模型, 然后就可以不输入命令, 直接按回车, 调试器会重复前面的 **step** 命令, 它如下所示:

```
(sldebug @0:0 'vdp/x1') : step
U1 = [0]
Y1 = [2]
(sldebug @0:1 'vdp/Out1') :
U1 = [2]
(sldebug @0:2 'vdp/x2') :
U1 = [0]
Y1 = [0]
(sldebug @0:3 'vdp/Out2') :
U1 = [0]
```

## 2. 关于模块索引

Simulink 的许多命令和信息都是通过模块索引来指代模块。模块索引的通用形式为 **s:b**, 其中 **s** 是一个标识当前模块在被调试模型中所处的系统的整数, 而 **b** 则是标识该系统中的第几个模块。例如 **0:1** 表示模型中系统 0 的模块 1。Simulink 提供了一个 **slist** 命令来显示模块中各个模块的索引。

```
>> slist

---- Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks, directFeed=0]
0:0   'vdp/x1' (Integrator)
0:1   'vdp/Out1' (Output)
0:2   'vdp/x2' (Integrator)
0:3   'vdp/Out2' (Output)
0:4   'vdp/Fcn' (Fcn)
0:5   'vdp/Product' (Product)
0:6   'vdp/Mu' (Gain)
0:7   'vdp/Scope' (Scope)
0:8   'vdp/Sum' (Sum)
```



因为虚拟模块对模型没有实质性的作用，所以在调试时，Simulink 不会为这些模块指定索引。

### 3. 访问 MATLAB 工作空间

读者不但可以在在调试命令提示符下输入调试命令，而且还可以输入任何有效的 MATLAB 命令。例如，读者可以在模型中设置一个断点，并把模型的时间和输出保存在 MATLAB 变量 tout 和 yout，然后就可以用下面的命令

```
(sldebug ...): plot(tout,yout);
```

来绘制一个输出—时间曲线。

如果用户试图访问一个名称和 Simulink 调试器命令名称的部分或者全部相同的变量，例如是 s，它和 step 命令的一部分相同。如果用户在调试器提示符下直接输入 s，就意味着输入了 step。但可以用下面的命令来显示变量 s 的值。

```
(sldebug...):eval('s')
```

## 7.2 增量运行模型

单步运行程序是几乎所有的调试器都具有的标准功能，Simulink 也不例外。Simulink 允许用户从一个模块跳转到另一个模块，从一个时间点到另一个时间点，从一个断点到另一个断点。用户可以选择表 7-1 中合适的仿真命令来推进仿真。

表 7-1 单步执行的调试器命令

命 令	推进仿真的方式
step	一个模块
next	一个时间步
continue	到下一个断点
run	仿真的终点，并忽略断点

### 1. 按模块单步执行

若想让仿真按模块单步执行，可以在命令窗口输入 step 命令，该命令执行当前的模块后停止，并加亮显示模型中的模块执行次序中的下一个模块。例如，图 7-2 显示了执行完第一个模块后的模块图表。

如果紧接着要执行的模块出现在子系统里，那么调试器就会打开子系统的图表并加亮显示这个模块。在执行完一个模块后，调试器在命令窗口显示模块的输入和输出，并显示调试命令提示符，提示符里会显示下一个要执行的模块的索引。例如，

```
(sldebug @0:0 'vdp/x1'): step
```

```
U1 = [0]
```

```
Y1 = [2]
```

```
(sldebug @0:1 'vdp/Out1'):
```

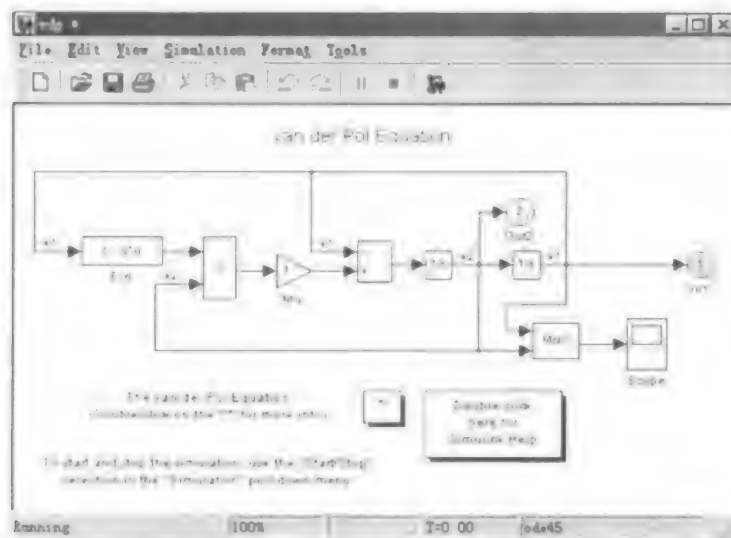


图 7-2 执行第一个模块时的模型图表

## 2. 按时间步单步执行

当用户执行完模型排序列表中的最后一个模块时，调试器把模型推进到下一个仿真时间步，并暂停在下一个仿真时间步要执行的第一个模块的开始处。为了指示用户处于仿真时间步的边界，调试器在 MATLAB 命令窗口显示当前的时间。例如，执行完 vdp 第一个时间步的最后一个模块后，会产生如下的输出。

```
(sldebug @0:8 'vdp/Sum') : step
U1 = [2]
U2 = [0]
Y1 = [-2]
[Tm=0.0001004754572603832 ] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/x1') :
```

读者可以按最小的时间步单步执行仿真，也可以按最大的时间步来执行。这时，要使用命令 `minor`。

`next` 命令执行当前仿真时间步中的其余模块。效果上，这个命令允许用户只通过一个命令直接跳到下一个仿真时间步，否则使用 `step` 则一个模块接一个模块执行，如果模型复杂就需要很多次命令了。`next` 命令在用户对当前时间步剩余的模块的行为没有兴趣时使用就十分方便了。将仿真推进到下一个仿真时间步后，调试器停留在执行顺序表中的第一个模块。例如

```
(sldebug @0:0 'vdp/x1') : next

[Tm=0.0006028527435622993 ] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/x1') :
```

3. 推进到下一个断点

`continue` 命令将仿真从当前的断点推进到下一个断点和仿真的终点两者中最先出现的一个。

如果用户确信整个仿真过程的余下部分没有感兴趣的事情发生，就可以使用 `run` 命令从当前仿真点无中断的运行到仿真的末尾。注意，中间设置的任何断点对 `run` 命令都不起作用。在仿真的终点，调试器将把控制权交给 MATLAB 命令行。这时如果用户想继续调试一个模型，就必须重新启动调试器，方法还是前面所讲到的。

## 7.3 设置断点

提供设置断点的能力也是调试器的标准功能之一，在前面一节讲过，`continue` 命令可以将仿真从一个断点运行到另一个断点，这样就提高了用户对仿真过程的控制能力。那在 Simulink 里用什么命令来设置断点呢？

Simulink 调试器允许用户设置的断点有两种类型：无条件的和条件的。无条件断点无论何种情况下，在仿真到达先前标记为断点的模块或时间步时就生效，也就是使仿真暂停。而条件断点只有在用户预先定义的条件发生的情况下，才会生效。表 7-2 列出了可以用来设置断点的调试命令以及它们各自的作用。

表 7-2 用于设置断点的调试命令

命 令	产生的仿真中断点的位置
<code>break &lt;gcb   s:b&gt;</code>	在模块索引 s:b 标识的模块的开始时
<code>bafter &lt;gcb   s:b&gt;</code>	在模块索引 s:b 标识的模块的结束时
<code>tbreak [t]</code>	在仿真时间步 t
<code>nanbreak</code>	在上溢或下溢 (NaN) 或者无穷大值 (Inf) 发生时
<code>xbreak</code>	当仿真到达决定仿真步长的状态
<code>zcbreak</code>	当过零点发生在仿真时间步之间时

### 7.3.1 非条件中断

1. 在模块处中断

`break` 命令让用户在模块的开始处设置断点，这样，在每一个时间步，当仿真运行至这个模块都会停住。

指定模块的方法既可以是使用模块索引，也可以是使用图形方法。使用图形方法就要将 Simulink 的模型窗口和 MATLAB 命令窗口结合起来。方法是首先在模型图表里用鼠标选中要设置断点的模块，然后在命令窗口调试命令提示符后输入下面的命令

```
(sldebug @0:0 'vdp/x1') : break gcb % 将断点设置在模型图表中选中的模块的开始处
```

Break point 0:4 'vdp/Fcn' installed.

(sldebug @0:0 'vdp/x1') :

用户也可以用它的模块索引来标识模块。按下面的格式输入命令：

```
>> break s:b
```

其中 s:b 是模块的索引，如果用户不清楚各模块的索引，可以用 `slist` 来查询。例如 vdp 模型中的 Fcn 模块的索引是 0:4。于是用下面命令也可以在模块的开始处设置断点。

```
>> break 0:4
```

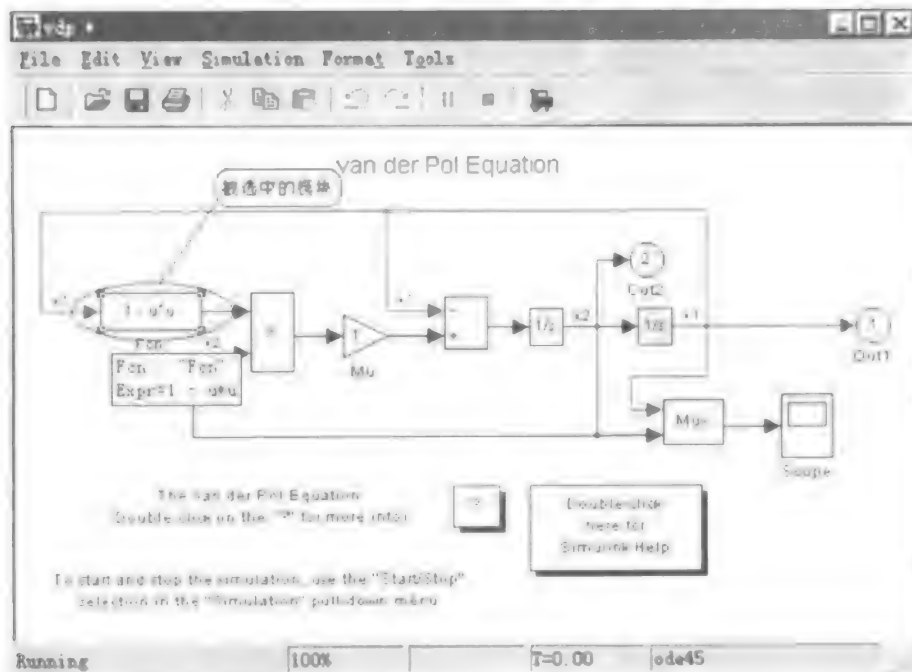


图 7-3 图形方法中选中模块示意

- ✎ 用户不能在虚拟模块设置断点。道理很简单，虚拟模块的作用仅仅是图形方面的：表示模型中计算模块的组合和关系。如果用户试图在虚拟模块设置断点，调试器将会给出警告。而 `slist` 命令也只列出了非虚拟模块的索引。

使用类似的格式，用户可以用 `bafter` 命令在非虚拟模块的结束处设置断点。

清除断点的命令是 `clear`，它可以清除在模块处设置的断点。使用的方法也有模块索引和图形两种，这些和 `break` 命令是很类似的。例如，

```
>> clear 0:4
```

或者，先在模型图表中选定 Fcn 模块，然后用 `gcb` 作为 `clear` 命令的输入参量。

```
>> clear gcb
```

注意，Simulink 调试器和许多高级语言清除断点的方法不一样，在那些高级语言里，在

已经设为断点的地方再把它设为断点，就会清除这些断点。但在 Simulink 调试器里情况就不是这样。例如，前面已经设置过 0:4 模块的断点，如果再使用 `break` 命令，并没有清除这个断点，调试器只是显示一个 'vdp/Fcn' 断点已经存在的信息。例如，

```
(sldebug @0:0 'vdp/x1') : break 0:4
```

```
Break point @0:4 'vdp/Fcn' is already installed.
```

## 2. 在时间步中断

若想在特定的时间步设置断点，用户可以使用命令 `tbreak`。这个命令只需要传入一个时间参量 `t` 用来指定要设置断点的时刻，使用它之后，当仿真运行到 `t` 所指定的时刻时，仿真就中断。例如，

```
(sldebug @0:0 'vdp/x1') : tbreak 5
```

```
(sldebug @0:0 'vdp/x1') : continue
```

```
Time break point found (tbreak) .
```

```
[Tm=5.051795155935553      ] **Start** of system 'vdp' outputs
```

注意参量 `t` 指定的时间是真正的时间，而不是仿真时间步的个数，如上例中就是指在 5 秒处设置断点，但调试器在运行时，却是在所有大于 `t` 的最接近的仿真时间步停止，所以仿真中断的时刻是 5.05……。

## 7.3.2 条件中断

条件中断和非条件中断的区别在于，条件中断下，模型被中断的具体时机的不确定性，即不改变中断设置，每次中断发生情况，也会不尽相同，它受到仿真运行具体情况的影响。而无条件断点则不同，例如，只要用户用 `break` 在某个模块设好了断点，无论仿真的输入如何改变，只要仿真运行到这个模块，中断都会发生，即不受执行状况的影响。Simulink 用于设置条件中断的命令有：`nanbreak`、`xbreak` 和 `zcbreak`。

### 1. nanbreak

`nanbreak` 命令用来设置在仿真中出现无限大的值时发生的中断。设置了这种中断，如果在仿真中计算出无穷大的数或者是超出运行仿真的机器的表示范围的数（上溢和下溢），调试器将会中断仿真。`nanbreak` 在查找模型的计算性错误时，非常有用。

### 2. xbreak

用 `xbreak` 状态可以设置这样的中断，如果模型使用步长可变换解法器，并且解法器遇到一个会限制解法器可以采用的步长的状态，调试器就中断仿真。这个命令在调试的模型出现需要更多的仿真步的情况时非常有用。

### 3. zcbreak

`zcbreak` 命令可以产生在出现过零点时发生的中断。这时，当 Simulink 检测到一个非采样过零点出现在模型里，或者是模型包括可能产生过零点的模块，调试器就暂停仿真。暂停

之后，会在命令窗口显示中断在模型中的位置、时间和过零点的类型（rising 还是 falling）。例如，在 `zerosxing` 演示模型开始执行前设置一个过零点中断。

```
>> sldebug zerosxing % 在调试器模式下，执行 zerosxing 模型
```

```
[Tm=0] **Start** of system 'zerosxing' outputs
```

```
(sldebug @0:0 'zerosxing/Sine Wave') : zcbreak
```

```
Break at zero crossing events is enabled.
```

然后用 `continue` 命令继续仿真

```
(sldebug @0:0 'zerosxing/Sine Wave') : continue
```

```
[Tm=0.34350110879329] Breaking at block 0:5
```

```
[Tm=0.34350110879329] Rising zero crossing on 3rd zcsignal in block 0:5 'zerosxing/Saturation'
```

此时，`zerosxing` 的模型图表就加亮显示出现中断的模块，见图 7-4。

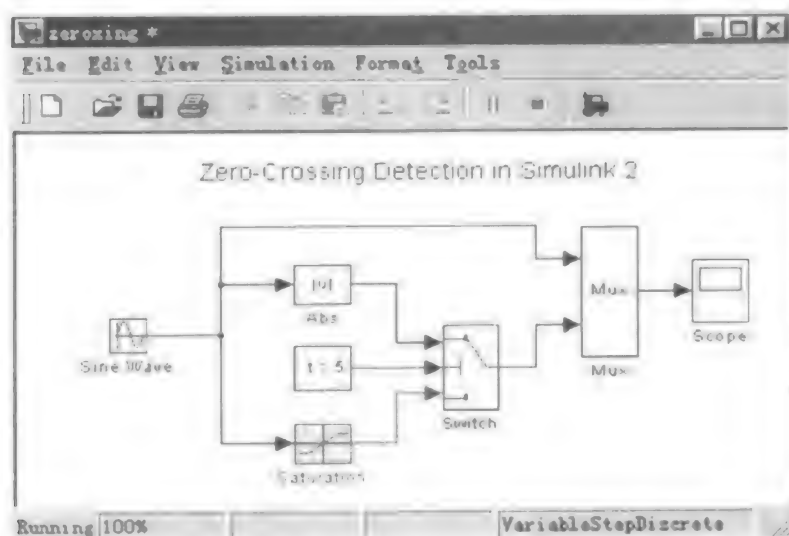


图 7-4 过零点中断示意

如果用户为一个没有能产生非采样过零点的模块的模型设置过零点中断，调试器会显示信息来告诉用户这一点。

## 7.4 显示仿真有关的信息

Simulink 调试器提供了许多命令允许用户在模型运行时，显示模块的状态、模块的输入输出和其他的信息。

7.4.1 显示模块的输入输出 I/O

调试器提供了三个命令来显示模块的 I/O，这些命令都可以显示指定模块的输入和输出，它们间的不同之处在于什么时候显示 I/O。这三个命令分别是：probe、disp 和 trace。

1. probe

probe 命令显示用户通过命令参量指定的模块的输入和输出，显示的位置是在 MATLAB 命令窗口。表 7-3 列出了 probe 命令的不同用法。

表 7-3 probe 命令的不同用法

命 令	描 述
probe	进入或者退出 probe 模式，在 probe 模式下，调试器显示用户在模型图表中所选择的任何模块的当前输入和输出。这时，在提示符后输入任何命令都会使调试器退出 probe 模式
probe gcb	显示被选中模块的输入和输出
probe s:b	显示由索引 s:b 所指模块的输入和输出

probe 命令可以方便的用于显示非当前执行模块的输入和输出。因为在调试时，比如，用 step 命令，调试器只会自动的显示最近执行完的模块的输入和输出，而 probe 命令可以让用户查询其他模块现有（最近时刻）的输入和输出。而 next 命令直接跳到下一个仿真时间步，执行过程中，调试器不显示模块的输入和输出，这时，就可以使用 probe 命令来检查模块的输入和输出。例如

```
(sldebug @0:5 'vdp/Product') : step
                                %仿真已经运行到模块 0:5，调试器显示了它的输入输出

U1 = [-3]
U2 = [0]
Y1 = [0]

(sldebug @0:6 'vdp/Mu') : probe 0:2
                                % 此时，想查询前面执行过的模块 0:2 的输入输出

I/O of 0:2 'vdp/x2':
U1 = [0]
Y1 = [0]

(sldebug @0:6 'vdp/Mu') : probe 0:6
                                % 也可以查询在当前仿真时间步尚未执行的模块 0:6
                                % 的输入和输出，这时显示的是该模块在前一个仿真步
                                % 的输入和输出，如果当前处于第一个仿真步，那显示
                                % 的就是它的初始输入和输出。

I/O of 0:6 'vdp/Mu':
U1 = [0]
Y1 = [0]

(sldebug @0:6 'vdp/Mu') : step
```

% 模块 0:6 执行后，它的输入输出。

```
U1 = [0]
```

```
Y1 = [0]
```

```
(sldebug @0:7 'vdp/Scope') : next
```

% 使用 next 命令将模型推进到下一个仿真时间步，  
显示器没有显示任何模块的输入和输出，而只显  
示 next 命令执行后仿真处于的时间。

```
[Tm=0.0001004754572603832 ] **Start** of system 'vdp' outputs
```

```
(sldebug @0:0 'vdp/x1') : probe 0:7
```

% 于是，用 probe 0:7 命令来查询模块 0:7 (scope)  
的输入和输出。

```
I/O of 0:7 'vdp/Scope':
```

```
U1 = [1.999999989905697 -0.0002009206312731412]
```

```
(sldebug @0:0 'vdp/x1') : probe 0:8
```

% 查询模块 0:8 的输入和输出，用户可以用 probe 进  
入 probe 模式，然后在模型图表中选择 sum 模块，调试  
器会自动在命令窗口显示 sum 模块的输入和输出。

```
I/O of 0:8 'vdp/Sum':
```

```
U1 = [1.999999989905697]
```

```
U2 = [0.0006027618857068085]
```

```
Y1 = [-1.99939722801999]
```

## 2. disp

disp 命令使调试器在仿真暂停的任何时候显示 disp 命令参数指定的模块的输入和输出。指定模块的方式既可以通过模块索引，也可以通过图形方式——先在模型图表选中模块，然后输入 gcb 作为 disp 命令的输入参量。用户可以用 undisp 命令将一个模块从调试器的显示模块列表中去掉，它同样支持模块索引和图形两种方式。

disp 命令的作用是用来跟踪特定的一个或多个模块在用户单步执行仿真时的输入和输出。使用 disp 命令指定这些模块后，调试器会在每一次单步执行停止时显示这些模型的输入输出。但要注意，由于使用 step 命令时，调试器会自动的显示当前执行模块的输入和输出，如果用户仅仅需要这样，就不必使用 disp 命令了。例如，紧接着前面的 vdp 模型的执行，用户需跟踪模块 Fcn 的输入输出。

```
(sldebug @0:0 'vdp/x1') : disp 0:5 % 把 0:5 模块加入到跟踪列表里
```

```
Display of block 0:5 'vdp/Product' installed.
```

```
(sldebug @0:0 'vdp/x1') : next
```

% 使用 next 命令，按理是不显示任何模块的输入输出的，但  
因为 0:5 模块需要被跟踪，所以调试器显示了它的输入和输出。

```
[Tm=0.01567417133261978 ] I/O of block 0:5 'vdp/Product'
```



```

U1 = [-2.999032582108157]
U2 = [-0.03062158372261323]
Y1 = [0.0918351272998699]
[Tm=0.01567417133261978    ] **Start** of system 'vdp' outputs

```

```

(sldebug @0:0 'vdp/x1') :
[Tm=0.07847133212035928    ] I/O of block 0:5 'vdp/Product'
U1 = [-2.977233198865903]
U2 = [-0.1397561358249672]
Y1 = [0.4160866073233047]
[Tm=0.07847133212035928    ] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/x1') : udisp 0:5
                        % 将模块 0:5 从显示模块列表中去掉。

```

注意如果使用 `step` 命令单步执行，使用 `disp` 的效果并不明显。

### 3. trace

`trace` 命令使调试器显示指定的模块的输入输出，使用了这个命令的模块，Simulink 无论何时对它们进行估值，调试器都会显示它们输入和输出。它让用户不中断仿真就可以获得一个模块输入和输出的完整记录。与别的显示信息命令一样，`trace` 同样支持模块索引作为参量和 `gcb` 作为参量的图形方式这两种格式。如果要把一个模块从调试器的 `trace` 列表去掉，可以使用 `untrace` 命令，它同样支持前面常提的两种格式。例如，

```

[Tm=0.07847133212035928    ] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/x1') : tbreak 1
                        % 在 1 秒钟设置一个断点
(sldebug @0:0 'vdp/x1') : continue
                        % 将仿真停止在前面所设置的断点前
.....
[Tm=1.27878459829406       ] **Start** of system 'vdp' outputs

(sldebug @0:0 'vdp/x1') : trace 0:4
                        % 将模块 0:4 加入到调试器的 trace 列表中。
Trace of block 0:4 'vdp/Fcn' installed.
(sldebug @0:0 'vdp/x1') : tbreak 1.5
                        % 在 1.5 秒处设置断点
(sldebug @0:0 'vdp/x1') : continue
                        % 仿真将运行至 1.5 秒左右处停止，这期间模块 0:4
                        % 的输入和输出将被显示。
[Tm=1.27878459829406       ] I/O of block 0:4 'vdp/Fcn'

```

```

U1 = [1.26874167682038]
Y1 = [-0.6097054425009887]
Time break point found (tbreak) . % 在时间断点处停止
[Tm=1.67878459829406      ] I/O of block 0:5 'vdp/Product'
U1 = [0.3223054432827113]
U2 = [-1.319256553049167]
Y1 = [-0.4252035681341334]
[Tm=1.67878459829406      ] **Start** of system 'vdp' outputs

```

从上面的例子可以看出，在模型运行中（从一个断点运行到另一个时间断点），仿真不用停止就可以显示模块 0:4 在各个仿真时间步的输入输出值，至于其中用了两个中断，则是为了使要显示执行的结果不要太长，方便举例。

#### 7.4.2 显示代数环信息

atrace 命令可以使调试器显示模型中的代数环在每个时间步内被求解的有关信息。atrace 命令只有一个参量，它的用法如表 7-4 所示。

表 7-4 atrace 命令的使用格式

命 令	显示的信息
atrace 0	没有信息显示
atrace 1	环路变量的解，求解环路所需的迭代次数，以及估计的解的误差
atrace 2	通 atrace 1
atrace 3	atrace 2 所显示的信息加上求解环路的夹可比（Jacobian）矩阵
atrace 4	atrace 3 所显示的信息加上环路变量在每次迭代的瞬时解

#### 7.4.3 显示系统状态

states 调试命令列出在 MATLAB 命令窗口列出系统状态的当前值。例如，下面的命令序列显示了 Simulink 的反弹球示例模型（bounce）在第一、第二个仿真时间步的状态值。

```

sldebug bounce
[Tm=0                      ] **Start** of system 'bounce' outputs
(sldebug @0:0 'bounce/Position') : states
                                % 显示状态值
Continuous state vector (value,index,name) :
10                               0 (0:0 'bounce/Position')
15                               1 (0:5 'bounce/Velocit')
(sldebug @0:0 'bounce/Position') : next
[Tm=0.01                     ] **Start** of system 'bounce' outputs
(sldebug @0:0 'bounce/Position') : states

```

```

                                % 状态值的显示格式
Continuous state vector (value,index,name) :
    10.1495095                0 (0:0 'bounce/Position')
    14.9019                   1 (0:5 'bounce/Velocity')

```

#### 7.4.4 显示积分信息

ishow 命令将锁牢积分信息的显示。当它被使能时，这个选项使调试器显示它每进行一时间步或者遇上约束仿真时间步大小的状态时的时间信息。对于前面一种情况，调试器显示仿真时间步的大小，例如，

```

(sldebug @0:1 'bounce/Bouncing Ball Display') : next
                                % 对于第一种情况
[Tm=0.03                        ] Step of 0.01 was taken by integrator
[Tm=0.03                        ] **Start** of system 'bounce' outputs
                                % 对于第二种情况，调试器显示限制仿真时间步大小
                                的状态名和具有该状态的模块名。
(sldebug @0:0 'bounce/Position') : ishow
Display of integration information is enabled.
(sldebug @0:0 'bounce/Position') : next
[Ts=0.02                        ] Integration limited by 1st state of block 0:0 'bounce/Position'
                                % 对于第二种情况
[Tm=0.02                        ] Step of 0.01 was taken by integrator
[Tm=0.02                        ] **Start** of system 'bounce' outputs

```

## 7.5 显示模型的信息

调试器除了能显示关于仿真的信息外，还可以显示关于模型的信息。它们包括：显示模型中模块的执行次序表，显示模块，显示模型中的非虚拟系统和非虚拟模块，显示潜在的具有过零点的模块，显示代数环以及显示调试器的设置。下面就对它们一一进行介绍。

### 1. 显示模型中模块的执行顺序列表

我们知道 Simlink 要在模型运行开始时，模型的初始化期间，确定模块执行的次序列表。在仿真期间，Simulink 保存这个按执行顺序排列的列表。这里不妨称为排序表。用户可以在任何时候在调试命令提示符后输入 slist 命令来显示排序表，显示的列表包括了各模块的索引。实际上，模块索引就是按执行次序来排列的。例如，显示 vdp 模型的排序列表。

```

(sldebug @0:0 'vdp/x1') : slist

```

```
---- Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks, directFeed=0]
```

```
0:0   'vdp/x1' (Integrator)
0:1   'vdp/Out1' (Outport)
0:2   'vdp/x2' (Integrator)
0:3   'vdp/Out2' (Outport)
0:4   'vdp/Fcn' (Fcn)
0:5   'vdp/Product' (Product)
0:6   'vdp/Mu' (Gain)
0:7   'vdp/Scope' (Scope)
0:8   'vdp/Sum' (Sum)
```

由于只有非虚拟模块才会按执行次序排序，所以 `slist` 命令的另一个作用就是显示模型中的非虚拟模块。

## 2. 显示模型中的非虚拟系统

`systems` 命令可以显示模型中所有非虚拟系统的一个列表。读者可能对非虚拟系统这个概念有疑问。粗略地讲，非虚拟系统和非虚拟模块的意思是差不多的。所谓的虚拟系统是指那些仅仅起到图形组织功能的系统，即这样的一些子系统，它们在模型图表尽管表示为子系统模块，但在仿真时 Simulink 把它内部包含的模块作为父系统的一部分。在 Simulink 模型里，只有根系统和条件子系统（触发或者使能子系统）才是真正的系统——非虚拟系统，而其他的所有子系统都是虚拟子系统，都不会出现在 `systems` 命令显示的系统列表里。例如，Simulink 的 `clutch` 示例模型里有许多的子系统，但 `systems` 只显示了根系统和两个触发子系统。

```
>> sldebug clutch % 在命令窗口输入
```

```
[Tm=0 ] **Start** of system 'clutch' outputs
(sldebug @0:0 'clutch/Clutch Pedal') : systems
```

```
0   'clutch'
1   'clutch/Locked'
2   'clutch/Unlocked'
```

注意，对于已经封装后的子系统，Simulink 将它们作为模块来处理，同样不作为非虚拟系统来显示。

## 3. 显示具有过零点的潜在模块

`zclist` 命令可以列出模型中所有可能会产生非采样过零点的模块的列表。例如，`zclist` 命令列出 `clutch` 模块的所有产生过零点的潜在模块。

```
(sldebug @0:0 'clutch/Clutch Pedal') : zclist
```

```
2:3   'clutch/Unlocked/slip direction' (Signum)
```

```

0:4   'clutch/Friction Mode Logic/Lockup Detection/Velocities Match' (HitCross)
0:10  'clutch/Friction Mode Logic/Lockup Detection/Required Friction for Lockup/Abs' (Abs)
0:12  'clutch/Friction Mode Logic/Lockup Detection/Required Friction for Lockup/Relational Operator'
      (RelationalOperator)
0:19  'clutch/Friction Mode Logic/Break Apart Detection/Abs' (Abs)
0:20  'clutch/Friction Mode Logic/Break Apart Detection/Relational Operator' (RelationalOperator)
0:24  'clutch/Unlocked' (SubSystem)
0:27  'clutch/Locked' (SubSystem)

```

#### 4. 显示代数环

`ashow` 命令加亮显示一个特定的代数环或者包含某个特定模块的代数环。加亮显示一个特定代数环的方法是 `ashow s#n`，其中，`s` 指包含该代数环的系统的索引，`n` 指代数环在该系统内的索引。如果要加亮显示包含特定模块的代数环，可以使用 `ashow gcb`，这个命令将会加亮显示用户在模型图表中当前选中的模块（即图形方式）。当然，用户也可以通过 `s:b` 这种模块索引格式来指定模块。如果要清除代数环的加亮显示，可以使用 `ashow clear` 命令。

#### 5. 显示模块设置

`status` 命令用来显示模型中不同调试选项的设置情况，比如是否设置了条件断点等等。下面的命令序列显示了 `vdp` 模型的初始调试设置。

```

sldebug vdp
[Tm=0                               ] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/x1') : status
Current simulation time                : 0 (MajorTimeStep)
Default command to execute on return/enter : ""
Stop in minor times steps              : disabled
Break at zero crossing events           : disabled
Break when step size is limiting by a state : disabled
Time break point                       : disabled
Break on non-finite (NaN,Inf) values    : disabled
Number of installed break points        : 0
Number of installed display points       : 0
Number of installed trace points        : 0
Display of integration information       : disabled
Algebraic loop tracing level            : 0

```

读者可以试着设置一个时间断点，再使用 `status` 命令显示调试器设置，会发现 `Time break point` 这一项的值变成了 `enabled`。

## 7.6 Simulink4.0 的图形调试工具

在 Smulink3.0 以前的版本, 对模型进行调试都是使用命令行命令, 经常使用 VC++ 等高级语言的读者可能觉得很习惯, 事实上也的确不是很方便。在 MathsWork 公司最新推出的 Smulink4.0 里, 最新增加了一个图形调试工具——Simulink Debugger。这一节就对它进行简单的介绍。

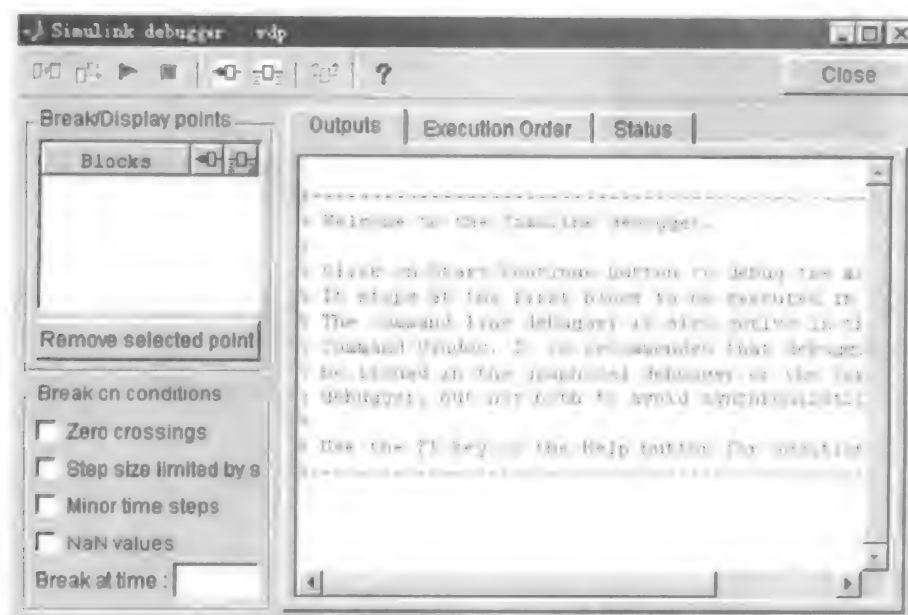


图 7-5 Simulink Debugger

读者可以通过 Tools 菜单下的 Simulink Debugger 命令来打开调试器, 也可以通过工具栏上的快捷按钮 (爬虫状, 可以将鼠标移到按钮上, 在 Simulink 模型窗口底部的状态栏可以看到当前按钮的说明信息) 来打开。图 7-5 就是对 vdp 演示模型进行调试时, 调试器打开后的样子。

整个调试器窗口分为两个部分, 左边的窗口用于显示模型的信息, 它分为三个页面: Outputs, Execution Order 和 Status。在调试器刚被打开时, 这些页面什么也不显示, 因为仿真的调试还没有开始。开始调试, 可以按工具栏上的快捷按钮 (向右的箭头)。

一旦开始了调试, 右边的信息显示窗口就会显示各自的信息。例如, Execution Order 页面就显示 vdp 模型中各模块的更新次序 (排序表), 同时也给出了各个模块的模块索引, 或者说是模型中的非虚拟模块列表, 这个信息在模型调试中不会变化。而 Status 页显示的信息是指调试器的当前设置, 它的作用和在命令行输入 status 命令是一样的。这一页显示的信息会随着用户的操作而变化, 例如用户设置新的断点, 都会对它有影响。

而 Outputs 页则用于调试命令执行后, 输出结果的显示, 它和使用命令行命令调试时, MATLAB 命令窗口的作用一样。图 7-6 是 vdp 模型在开始调试时, 调试器的 Outputs 页的输出。

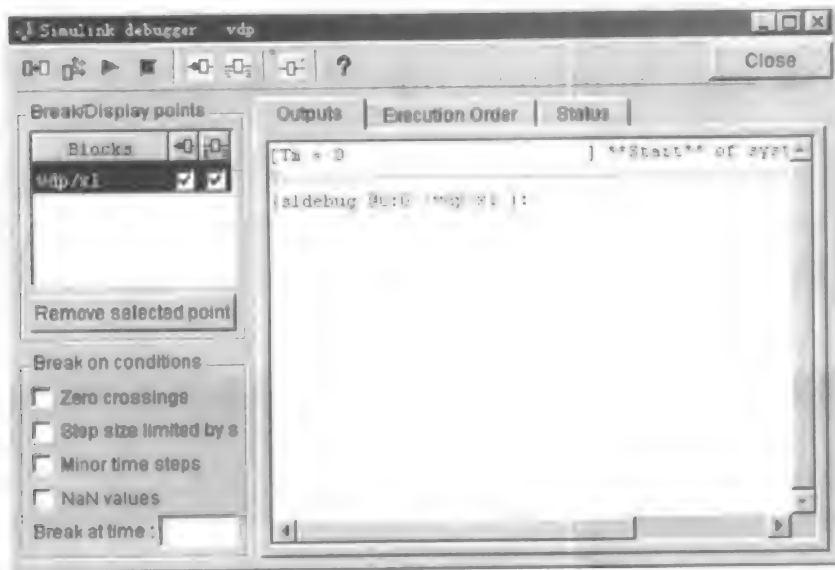









图 7-6 Output 页的输出

调试窗口的左边部分和工具栏的快捷按钮则是一些调试操作的图形命令。工具栏上的按钮的功能描述如下：

- (1)  运行仿真到下一个模块。
- (2)  运行仿真到下一个时间步的开始。
- (3)  开始或继续运行仿真。
- (4)  终止模型的调试。
- (5)  在当前所选择模块运行前设置断点。
- (6)  在所选择模块被执行时，显示它的输入和输出，即设置显示点。
- (7)  显示所选择模块当前仿真时间步的输入和输出。

关于这些操作的意义，请读者查看本章的前面几节。

这里要请读者注意，无论是设置断点，还是设置显示点，操作时都要回到模型窗口，先选中某个模块，然后在进行相应的操作，也就是采用前面所说的图形操作方式。

对于设置了断点或者是显示点的模块，调试器会自动的把它们模块名称添加到 **Breaks/Display points** 列表框。

Simulink 调试器还在窗口的左下端提供了几个检查框选项让用户进行条件中断点的设置，它们是：

- (1) **Zero crossings**，选中它表示遇到过零点检测时产生中断点；
- (2) **Step size limited by state**，选中它表示在步长受状态限制时产生中断点；

(3) Minor steps, 选中它, 则使仿真进入最小时间步模式;

(4) NaN values, 选中它表示如果在仿真时一个计算值为无限大或者超出了机器表示范围时产生中断点。

关于这些设置的详细描述也请读者查看前面的章节。

调试窗口左边最下端的 Break at times 编辑框的作用是为了让用户设置在某个时间步设置断点, 用户只须在里面输入要设置断点的实际时间, 注意不是仿真的步数。

## 7.7 调试命令使用参考

为了方便读者查询 Simulink 的各种调试命令, 在本章的最后给出这些命令的使用参考, 表 7-5 列出了这些命令。其中, 是否可重复这一列, 表示该命令是否可以通过用回车键(return)来重复执行。例如, 在调试命令提示符后输入 step 命令后, 就可以通过按回车键来重复执行 step 命令, 这省去不少命令输入的工作。在表 7-5 就详细地列出了每个命令的目的、语法、参量和说明信息。

表 7-5 调试命令列表

命 令	简 写 形 式	是否可重复	描 述
ashow	as	否	显示代数环
atrace	at	否	设置代数环的跟踪等级
bafter	ba	否	插入一个在模块执行后发生的断点
break	b	否	插入一个在模块执行前发生的断点
bshow	bs	否	显示特定的模块
clear	cl	否	清除一个设置在模块的断点
continue	c	是	继续进行仿真
disp	d	是	显示一个模块的输入输出
help	h	否	显示调试器命令的帮助信息
ishow	l	否	使能或禁止积分信息的显示
minor	m	否	使能或禁止最小步长模式
nanbreak	na	否	设置或清除无穷大值断点
next	n	是	跳到下一个时间步的开始
probe	p	否	显示一个模块的输入输出
quit	q	否	退出仿真
run	r	否	运行完剩余的仿真
slist		否	列出模型的非虚拟模块和模块执行顺序表
states	state	否	显示模型的当前值
status	stat	否	显示模型的调试器选项设置
step	s	是	步进到下一个模块



(1) **ashow**

目的 显

语法 `ashow <gcb |`

参量 s:b 系统索引是 s 模块索引是

**gcb** 当前模块;

s#n 系统 s 标号

**clear** 清除代数环加亮显示的开

ashow gcb 或者 ashow s:b 加亮显示

参见 atrace, slist

(2) atrace

## 目的 设

语法 **atrace level**

参量 level 跟踪等

说明 `atrace` 命令设置仿真的代数环跟踪等级,

参见 systems, states

命令	
----	--

目的 插

1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 26

语法	<b>bafter</b> <b>gcb</b> <b>bafter</b> <b>s:b</b>
参量	<b>s:b</b> 索引为 s 的系统内的索引为 b 的模块; <b>gcb</b> 当前模块。
说明	<b>bafter</b> 命令插入由参量指定的模块执行后的断点, 当该模块执行后, 仿真在此中断。
参见	<b>break</b> , <b>xbreak</b> , <b>tbreak</b> , <b>nanbreak</b> , <b>zcbreak</b> , <b>slist</b> 。

(4) **break**

目的	插入模块执行前生效的断点。
语法	<b>break</b> <b>gcb</b> <b>break</b> <b>s:b</b>
参量	<b>s:b</b> 索引为 s 的系统内的索引为 b 的模块; <b>gcb</b> 当前模块。
说明	<b>break</b> 命令插入由参量指定的模块执行前的断点, 在该模块被执行前, 仿真在此中断。
参见	<b>bafter</b> , <b>tbreak</b> , <b>xbreak</b> , <b>nanbreak</b> , <b>zcbreak</b> , <b>slist</b> 。

(5) **bshow**

目的	显示指定的模块。
语法	<b>bshow</b> <b>s:b</b>
参量	<b>s:b</b> 索引为 s 的系统内的索引为 b 的模块。
说明	这个命令打开包含指定模块的模型窗口并且选中该窗口。
参见	<b>slist</b> 。

(6) **clear**

目的	清除一个模块断点。
语法	<b>clear</b> <b>gcb</b> <b>clear</b> <b>s:b</b>
参量	<b>s:b</b> 索引为 s 的系统内的索引为 b 的模块; <b>gcb</b> 当前模块。
说明	这个命令清除设置在指定模块处的断点, 无论是执行前还是执行后。
参见	<b>bafter</b> , <b>slist</b> 。

(7) **continue**

目的	继续执行仿真。
语法	<b>continue</b>
说明	<b>continue</b> 命令从当前仿真中断点继续执行仿真, 仿真持续执行到它遇上另一个断点和仿真的结束时间。
参见	<b>run</b> , <b>stop</b> , <b>quit</b> 。

(8) **disp**

目的	当仿真停止时显示一个模块在停止时刻的输入输出。
语法	<b>disp</b> <b>gcb</b>

**disp s:b**  
**disp**  
**参量** s:b 索引为 s 的系统内的索引为 b 的模块;  
gcb 当前模块。  
**说明** 这个命令将指定模块注册为 display 点。无论何时仿真暂停, 调试器会在 MATLAB 命令窗口显示所有 display 点在暂停时刻的输入输出。调用无参量的 disp 命令显示所有 display 点的列表, 而 undisp 命令将指定模块从 display 点列表中去掉。  
**参见** undisp, slist, probe, trace。

## (9) help

**目的** 显示调试器命令的帮助信息。  
**语法** help  
**说明** help 命令在命令窗口显示调试器命令的一个列表, 该列表包含每个命令的语法和主要描述。

## (10) ishow

**目的** 使能或禁止积分信息的显示。  
**语法** ishow  
**说明** 这个命令会锁牢仿真过程中积分信息的显示。  
**参见** atrace。

## (11) minor

**目的** 使能或禁止最小步长模式。  
**语法** minor  
**说明** minor 命令使调试器进入或退出最小步长模式, 在最小步长模式里, step 命令将在一个最小步内单步执行一个模块。在执行完模块排序列表中的最后一个模块后, 如果下一个最小步仍然完全处在当前的主时间步范围内, step 命令会把仿真推进到下一个最小时间步, 否则, step 命令把仿真推进到下一个主时间步的第一个最小时间步。minor 命令提供了一个局部细化每个主时间步长的能力。  
**参见** step。

## (12) nanbreak

**目的** 设置或清除非有限值断点。  
**语法** nanbreak  
**说明** nanbreak 命令使调试器无论何时遇到一个非有限值 (上溢或下溢——超出机器的表示范围, 或者无穷大的数值), 都中断仿真。如果非有限值断点已经设置, 那么 nanbreak 命令将清除这个断点。  
**参见** break, bafter, xbeak, tbreak, zebreak。

## (13) next

**目的** 单步执行到下一个时间步。  
**语法** next  
**说明** next 命令执行当前时间步内所有将要执行的模块, 然后停止在下一个仿真时间步的开始。执行 next 命令后, 调试器加亮显示下一个时间步内将要执行的第一

个模块，并且显示下一个时间步的时间。

参见 `step`。

#### (14) `probe`

目的 显示一个模块的输入和输出。

语法 `probe[<s:blgcb>]`

参量 `s:b` 索引为 `s` 的系统内的索引为 `b` 的模块；

`gcb` 当前模块。

说明 不带任何参量的 `probe` 命令将使调试器进入 `probe` 模式。在 `probe` 模式下，调试器可以显示用户在模型图表中选中的任何模块的输入和输出。退出 `probe` 模式，只要输入任何命令即可。`probe gcb` 使调试器显示当前选中的模块的输入输出，`probe s:b` 则显示由索引 `s:b` 指定的模块的输入和输出。

参见 `disp`, `trace`。

#### (15) `quit`

目的 退出仿真。

语法 `quit`

说明 `quit` 命令终止当前仿真。

参见 `stop`。

#### (16) `run`

目的 运行仿真至结束。

语法 `run`

说明 `run` 命令从当前的中断点运行仿真至它的结束时间。它忽略断点和 `display` 点，即不会中断也不会显示 `display` 点的输入和输出。

参见 `continue`, `stop`, `quit`。

#### (17) `slist`

目的 列举模型中的非虚拟模块。

语法 `slist`

说明 `slist` 命令列出正在调试的模型中的非虚拟模块列表，该列表显示每个模块的名称和模块索引。

参见 `systems`。

#### (18) `states`

目的 显示当前的状态值。

语法 `states`

说明 `states` 命令显示模型当前状态的一个列表，列表的内容包括状态的取值，状态所属模块的索引和名称。

参见 `ishow`。

#### (19) `systems`

目的 列出模型中的非虚拟系统。

语法 `systems`

说明 `systems` 命令在 MATLAB 命令窗口显示模型中的非虚拟系统，列表会给出系统

的名称和它在模型中的索引，这个索引就是用来确定模块索引 s:b 中的 s。

参见 `slist`。

#### (20) `status`

目的 显示当前调试选项。

语法 `status`

说明 `status` 命令显示了当前的调试选项。

参见 `slist`。

#### (21) `step`

目的 单步执行到下一个模块。

语法 `step`

参量 `step` 命令执行当前仿真步下一个要执行的模块（此时在模型图表中加亮显示的模块）。执行完 `step` 命令后，调试器在模型图表中加亮显示下一个要执行的模块以及它的输出信号线，此外，调试器还把这个模块的索引和名称作为调试命令提示符的一部分加以显示。

参见 `next`。

#### (22) `stop`

目的 终止仿真。

语法 `stop`

说明 `stop` 命令将终止仿真，调试器把控制权交还 MATLAB 命令窗口，即调试命令提示符消失。

参见 `run`, `stop`, `quit`。

#### (23) `tbreak`

目的 设置或者清除一个时间断点。

语法 `tbreak t`

`tbreak`

参量 `t` 要设置断点的时间值。

说明 `tbreak t` 命令在指定的时间步设置一个断点，如果一个断点已经存在与指定时间，那该命令将会清除这个断点。如果用户没有指定时间，那么 `tbreak` 命令在当前时间步设置一个断点。注意这个命令的参量 `t` 是时间值，而调试器设置断点时是在按时间步来设置的，一般是在大于时间 `t`，并最接近 `t` 的仿真时间步设置断点。

参见 `break`, `bafter`, `xbreak`, `nanbreak`, `zcbreak`。

#### (24) `trace`

目的 在模块每次执行时显示模块的输入输出。

语法 `trace s:b`

`trace gcb`

参量 `s:b` 索引为 `s` 的系统内的索引为 `b` 的模块；

`gcb` 当前模块。

说明 `trace` 命令将指定的模块注册为 `trace` 点，调试器在注册模块每次执行时都显示

它的输入和输出。

参见 run, stop, quit。

#### (25) undisp

目的 将一个模块从调试器的 display 点列表中去掉。

语法 undisp gcb

undisp s:b

参量 s:b 索引为 s 的系统内的索引为 b 的模块；

gcb 当前模块。

说明 undisp 命令将指定模块从调试器的 display 点列表中去掉。

参见 disp, slist。

#### (26) untrace

目的 将一个模块从调试器的 trace 列表中去掉。

语法 untrace s:b

untrace gcb

参量 s:b 索引为 s 的系统内的索引为 b 的模块；

gcb 当前模块。

说明 untrace 命令将指定模块从调试器的 trace 点列表中去掉。

参见 trace, slist。

#### (27) xbreak

目的 在调试器遇到限步长状态时中断。

语法 xbreak

说明 xbreak 命令使调试器进入 xbreak 模式，即在遇到一个限制解法器采取的仿真时间步长的状态时，调试器会中断模型的执行。如果 xbreak 模式已经开启，那么 xbreak 命令将关掉这个模式。

参见 break, bafter, zcbreak, tbreak, nanbreak。

#### (28) zcbreak

目的 设置非采样过零点事件断点。

语法 zcbreak

说明 zcbreak 命令使调试器进入过零点中断模式，即调试器在非采样过零点事件发生时中断仿真，如果过零点中断模式已经开启，那么 zcbreak 命令将关闭该模式。

参见 break, bafter, xbreak, tbreak, nanbreak, zclist。

#### (29) zclist

目的 列出可能包含非采样过零点的模块。

语法 zclist

说明 zclist 命令在 MATLAB 命令窗口，显示有可能发生非采样过零点事件的模块的列表。

参见 zcbreak。



## 第八章 模块使用参考

模块是 Simulink 的基石，这一章将给出 Simulink 中部分模块的使用说明，它们按照字母顺序排列。对于每个模块，该使用参考包含的信息有下列内容。

- (1) 模块名称和包含该模块的模块库；
- (2) 模块的目的；
- (3) 模块使用的说明信息；
- (4) 模块对话框的参数；
- (5) 模块特性，包括下面所述的一部分或者全部：

➤ **Direct Feedthrough**——模块或者它的某个端口是否具有直接馈入。

➤ **Sample Time** ——模块的采样时间如何决定，是否是由模块本身，还是由它的驱动模块或者被它驱动模块继承。

➤ **Scalar Expansion** ——标量值是否可以扩展为向量。有些模块会把标量输入或者参数扩展为向量。

➤ **States** ——离散和连续状态数。

➤ **Vectorized** ——模块是否能接收或者产生向量信号。

➤ **Zero Crossings** ——模块是否检测状态事件。

### 1. Abs 模块

目的 输出模块输入信号的绝对值。

库 Math

说明 该模块把模块的输入信号的绝对值作为输出。

数据类型 模块接收类型为 `double` 的实数或者复数信号，输出一个 `double` 类型的实数。

特性

- |                             |         |
|-----------------------------|---------|
| ➤ <b>Direct Feedthrough</b> | 是       |
| ➤ <b>Sample Time</b>        | 从驱动模块继承 |
| ➤ <b>Scalar Expansion</b>   | N/A     |
| ➤ <b>Zero Crossings</b>     | 是，检测零   |

### 2. Algebraic Constraint 模块

目的 将输入信号约束为 0。

库 Math

说明 Algebraic Constraint 模块将输入信号  $f(z)$  约束为 0 并且输出一个状态变量  $z$ 。模块输出使输入变为零所需的状态值，输出信号必须通过某种回路来影响输出，这样就可以被用来求解代数方程。

缺省情况下，Initial guess 参数值为 0，可以通过把它的值设置为接近方程解的值，来提



高代数环解法器的效率。

数据类型      输入和输出 double 类型的实数信号。

参数  
Initial guess    解的初始猜测，通常是 0。

特性

- Direct Feedthrough      是
- Sample Time              从驱动模块继承
- Scalar Expansion        否
- Vectorized                是
- Zero Crossings          否

3. Band-Limited White Noise 模块

目的              将白噪声引入连续系统。

库                Sources

说明              Band-Limited White Noise 模块生成适合在混合系统和连续系统使用的正态分布随机数。这个模块和随机数模块的基本区别在于 Band-Limited White Noise 模块能以一个设定的速率产生输出，这个采样时间和噪声的相关时间有关。理论上，连续白噪声具有 0 相关时间，平功率谱密度，以及无穷大的方差。但在实际系统中，白噪声是不可能产生的，但它是一个有用的理论近似。在 Simulink，用户通过用一个相关时间小于系统的最短时间常数的随机序列来模拟白噪声的影响。而 Band-Limited White Noise 模块产生的随机序列的相关时间为模块的采样速率。为了更精确地仿真，可以把相关时间设置成比系统的最快速率还要小，按如下公式设定

$$t_c = \frac{2\pi}{100f_{\max}}$$

其中， $f_{\max}$  是以 rad/sec 为单位的系统带宽。

参数

- Noise power              白噪声功率谱密度的高度，缺省值为 0.1。
- Sample time              噪声的相关时间，缺省值为 0.1。
- Seed                      随机数发生器的初始种子，缺省值为 23341。

特性

- Sample Time              离散
- Scalar Expansion        Noise power 和 Seed 参数，以及输出具有标量扩展
- Vectorized                是
- Zero Crossings          否

4. Bus Selector 模块

目的              从输入总线选择信号。

库                Signals&Systems

**说明** Bus Selector 模块从 Mux 模块或者其他的 Bus Selector 模块接收输入, 这个模块有一个输入端口。而输出端口的数目依赖于 Muxed output 检查框的状态。如果它被选中, 则表明所有的输出信号被联合, 所以模块仅有一个输出端; 否则, 每一个被选中的信号都有一个输出端口与之对应。通常 Simulink 在模型图表中隐藏 Bus Selector 模块的名称。

**数据类型** 它可以接收和输出任何类型的实数和复数信号。

**参数**

#### Signals in the bus

Signals in the bus 列表显示了在输入总线的信号。使用 Select>>按钮从总线列表里选择输出信号。

#### Selected signals

Selected signals 列表显示输出信号。可以使用 Up、Down 和 Remove 按钮将信号排列次序, 当信号次序变化时, 端口的链接状况将不会改变。如果列在 Selected signals 列表的输出信号不是 Bus Selector 模块, 该信号的名称将会出现“???”。

输出端口的标签自动由模块来设置, 除非 Muxed output 检查框被选中, 当用户试图改变标签时, 将会出现错误信息。

### 5. Chirp Signal 模块

**目的** 生成频率增加的正弦信号。

**库** Sources

**说明** Chirp Signal 模块生成了一个频率以一个相对于时间的线性速率增加的正弦波。这个模块被用于非线性系统的频谱分析。模块生成标量或者向量信号。

**数据类型** 输出 double 类型的实数信号。

**参数**

#### Initial frequency

信号的初始频率, 可以被设定为标量或向量信号。缺省值为 0.1Hz。

#### Target time

频率达到目标频率的时间, 可以是标量也可以是向量值, 在这个时间后, 频率将会以相同速率变化。缺省值是 100s。

#### Frequency at target time

在目标时间的信号频率, 可以是标量也可以是向量值。缺省值为 1Hz。

**特性**

- Sample Time 连续
- Scalar Expansion 参数
- Vectorized 是
- Zero Crossings 否

### 6. Clock 模块

**目的** 显示或者提供仿真时间。

库	Sources
说明	Clock 模块在每一个仿真步输出当前的仿真时间。在其他的模块需要仿真时间时，这个模块就非常有用。如果是在离散系统里，就使用 Digital Clock 模块。
数据类型	输出 double 类型的实数信号。
参数	<p>Display time 使用 Display time 检查框使模块在图标显示当前的仿真时间。</p> <p>Decimation Decimation 参数值是模块获得更新所需的步数，可以为任意的正整数。例如当 Decimation 为 1000 时，对于固定积分时间步为 1（ms），时钟将每隔一秒钟更新。</p>
特性	<ul style="list-style-type: none"><li>➤ Direct Feedthrough</li><li>➤ Sample Time 连续</li><li>➤ Scalar Expansion N/A</li><li>➤ Vectorized 否</li><li>➤ Zero Crossings 是</li></ul>

7. Combinatorial Logic 模块

目的	实现真值表。
库	Math
说明	Combinatorial Logic 模块实现一个标准的真值表，用于对可编程逻辑阵列（PLA）、判决电路、决策表和其他的布尔表达式进行建模。用户可以将这个模块和 Memory 模块结合来实现有限状态机，或者 flip-flop 器件。

用户可以设定一个矩阵，定义所有可能模块的输出作为真值表。矩阵的每一行定义了不同输入结合下的输出。用户必须为每一个输入的结合设定输出。矩阵的列数是模块输出的数目，而矩阵的行数则是所有输入的组合，即为

$$2^{n\_input}$$

Simulink 从输入向量计算行数的下标，向量元素的值为 0 和 1。它把这个向量作为二进制数来处理，但因为矩阵下表从 1 开始，而不是 0，所以要在二进制数上加 1。

$$row\_index = 1 + u(m) * 2^0 + u(m-1) * 2^1 + \dots + u(1) * 2^{m-1}$$

数据类型	模块接收布尔类型或者 double 类型的实数信号，并且输出和输入相同的类型。真值表的元素类型可以是布尔或者 double 类型。如果是 double 类型，它们可以为任何值。如果真值表元素类型和输出信号的类型（输入信号）不同，Simulink 在计算前将真值表转换为输出的类型。
参数	Truth table

输出的矩阵，其中的每一列对应输出向量的一个元素，而每一行对应一个输入真值的组合。

#### 特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承
- Scalar Expansion 无
- Vectorized 是，输出信号的宽度是真值表参数的列数
- Zero Crossings 否

### 8. Complex to Magnitude-Angle 模块

**目的** Complex to Magnitude-Angle 模块和 Complex to Real-Imag 模块的作用很类似，都是对复数信号进行转换，前者转换成模长—辐角，后者转换成实部—虚部。

**库** Math

**说明** Complex to Magnitude-Angle 模块接收 double 类型复数信号，输出输入信号的模长和（或者）辐角，这取决于 Output 参数的设置。输入可以是向量信号，这时，输入信号相应的为向量。

**数据类型** 见说明

#### 参数

**Output**

决定模块的输出，有 MagnitudeAndAngle、Magnitude 和 Angle 三种取值。

#### 特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承
- Scalar Expansion 否
- Vectorized 否
- Zero Crossings 否

### 9. Configurable Subsystem 模块

**目的** 表示从任何用户设定模块库选择的模块。

#### 库

**说明** Configurable Subsystem 模块能够表示包含在一个指定模块库的任意模块，Configurable Subsystem 模块的对话框可以设置它所表示的模块和被表示模块的参数值。Configurable Subsystem 模块可以简化表示一类设计的模型的建立。例如，要对可以提供引擎选择的汽车进行建模，就可以在模型里使用 Configurable Subsystem 模块来表示引擎的选择。Configurable Subsystem 模块的外形取决于它所表示的模块。在初始情况下，它不代表任何模块，所以没有任何端口和图标显示。一旦选定库或者模块，子系统会显示图标和一系列端口，它们和所选库的输入和输出对应。Simulink 根据下面的法则来匹配库端口到 Configurable Subsystem 模块端口：

- (1) 将库中所有的具有唯一名称的端口，名称不变的显示在模块图标上；
- (2) 将所有具有相同名称的端口用同一个端口来显示；
- (3) 具有 Terminator/Ground 模块的模块库被选择时，将会终止未被使用的端口。

注意，这个模块不能表示非 I/O 端口，例如，受控子系统中的触发和使能端口。

数据类型 同所表示模块具有相同的数据类型。

参数

Configurable Subsystem 模块的对话框取决于所表示的模块。在初始状态，由于它不表示任何模块，所以只有一个参数——Library name 参数。

Library name

这个模块能表示模块所在的库的相对路径名。例如 Simulink/Math。注意，不能使用模块对话框来改变已存在 Configurable Subsystem 模块的 Library name，只能通过 set\_param 命令来进行。

- ✎ 如果一个模块增加或者被移去一个模块或者端口，那么使用这个模块的 Configurable Subsystem 模块就需要被重新建立。

Library name 的新的设定值，会产生一个 Block choice 域，一个 Open subsystems when selected 域，以及当前选中模块的参数。

Block choice

Configurable Subsystem 当前代表的模块。

Open subsystems when selected

选择这个选项将使 Simulink 打开所选中的没有封装的模块。

特性 同所代表的模块。

## 10. Constant 模块

目的 产生一个常数值。

库 Sources

说明 Constant 模块产生设定的、与时间无关的实数或者复数值。模块的输出可以是标量也可以是向量，取决于 Constant value 的长度。

数据类型 同 Constant value 参数的类型。

参数

Constant value

模块的输出，如果是一个向量，那么模块输出和设定值相同的向量信号。缺省值是 1。

特性

- Sample Time 常数
- Scalar Expansion 否
- Vectorized 是
- Zero Crossings 否

## 11. Coulomb and Viscous Friction 模块

目的 对在零点的不连续处建模，在别的地方具有线性增益。

库 Nonlinear

**说明** Coulomb and Viscous Friction 模块对库仑摩擦和粘滞摩擦建模。模块的模拟零点不连续但在其他地方具有线性增益。*offset* 对应库仑摩擦, *gain* 对应于粘滞摩擦。模块的实现可以是

$$y = \text{sign}(u) * (\text{Gain} * \text{abs}(u) + \text{Offset})$$

其中, *y* 是输出, *u* 是输入, Gain 和 Offset 是模块参数。

**数据类型** 接收和输出 double 类型的实数信号。

**参数**

Coulomb friction value

应用于所有输入值的偏移量, 缺省值是[1 2 3 0]。

Coefficient of viscous friction

在非零输入点的信号增益, 缺省值是 1。

**特性**

- Direct Feedthrough 是
- Sample Time 从驱动模块继承。
- Scalar Expansion 否
- Vectorized 是
- Zero Crossings 是, 在静态摩擦被克服的点。

## 12. Data Store Memory 模块

**目的** 定义数据存储。

**库** Signals&Systems

**说明** Data Store Memory 模块定义和初始化一个共享数据存储, 它是一个 Data Store Read 模块和 Data Store Write 模块存储区域。每一个数据存储必须由 Data Store Memory 模块定义。Data Store Memory 模块的位置决定了能访问它所定义的数据存储的 Data Store Read 模块和 Data Store Write 模块:

(1) 如果 Data Store Memory 模块处在顶层系统, 模型中任何位置的 Data Store Read 模块和 Data Store Write 模块都可以访问这个数据存储;

(2) 如果 Data Store Memory 模块处在一个子系统, 那么处在相同子系统或者在它下层系统的, 都可以访问所定义的数据存储。

可以通过在 Initial value 参数设定取值来初始化数据存储。值的大小决定了数据存储的大小。

**数据类型** 存储 double 类型的实数信号。

**参数**

Data store name

所定义的数据存储的名称, 缺省值是 A。

Initial value

数据存储的初始值, 缺省是 0。

**特性**

- Sample Time N/A

➤ Vectorized 是

### 13. Data Store Read (Write) 模块

目的 从数据存储读 (写)。

库 Signals&Systems

说明 Data Store Read 模块从一个命名的数据存储读取并把数据输出。数据是先前由 Data Store Memory 模块初始化的, 或者是由 Data Store Write 模块写到数据存储的数据。

数据类型 输出 double 类型的实数信号。

参数

Data store name

这个模块读取数据的 data store 的名称。

Sample time

采样时间, 控制何时写数据到 data store。缺省是-1, 表明采样时间是继承的。

特性

➤ Sample Time 连续或者离散

➤ Vectorized 是

### 14. Data Type Conversion 模块

目的 将输入信号转换为指定的数据类型。

库 Signals&Systems

说明 这个模块将输入信号转换为 Data type 参数设定的类型。输入可以是任何类型的实数或者虚数信号。

参数

Data type

设定要转换的类型。

Saturate on integer overflow

这个参数仅对整数输出适用。如果被选择, 这个选项将使模块在整数溢出时饱和。

特性

➤ Direct Feedthrough 是

➤ Sample Time 从驱动模块继承

➤ Scalar Expansion 参数可以

➤ Vectorized 是

➤ Zero Crossings 是, 检测溢出点

### 15. Dead Zone 模块

目的 提供一个零输出的区域。

库 Nonlinear

说明 Dead Zone 模块在设定的区域产生零输出, 称为死区。它的上下限由 Start of dead zone 和 End of dead zone 参数设定。而模块的输出取决于输入和死

区：

- (1) 如果输入在死区内，输出为零；
- (2) 输入大于等于上限，输出就是输入减去上限；
- (3) 输入如果小于或者等于下限，输出就是输入减去下限。

例如，图 8-1 所示的示例模型中，Dead Zone 模块的上、下限分别设为 0.5 和 -0.5，正弦波作为它的输入信号。

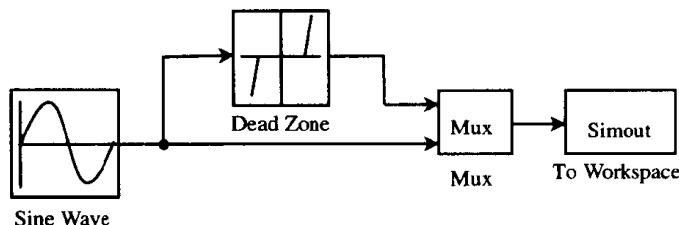


图 8-1 Dead Zone 模块示例模型

执行模型后，在 MATLAB 命令窗口用绘图命令 plot 就可以绘出输出曲线，它如图 8-2 所示。从图中可以看出，当正弦波的幅度处于  $[-0.5, 0.5]$  范围内时，Dead Zone 模块的输出为 0，而在其他区间，为了保证输出曲线的连续性，所以要按照前面说的 (2)、(3) 点来处理。

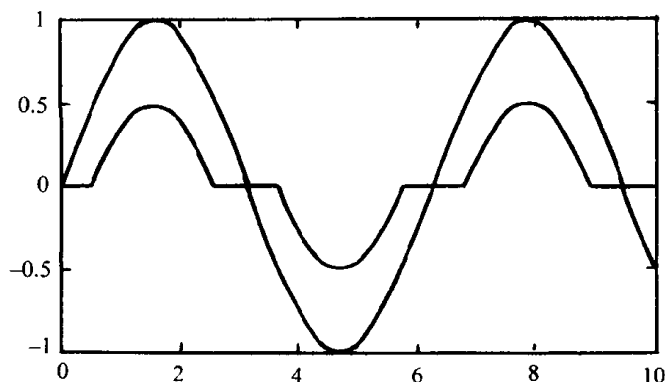


图 8-2 Dead Zone 模块的输出曲线

### 参数

#### Start of dead zone

设定死区的下限，缺省值为 -0.5。

#### End of dead zone

设定死区的上限，缺省值为 0.5。

### 特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承
- Scalar Expansion 参数
- Vectorized 是
- Zero Crossings 是，检测限达到的点



## 16. Derivative 模块

目的 输出输入信号的时间微分。

库 Continuous

说明 这个模块近似计算输入信号的时间微分，计算方法是通过求

$$\frac{\Delta u}{\Delta t}$$

在仿真开始时，假定输入为 0。计算的精度取决于仿真采用的时间步大小，时间步越小精度越高。但它和有连续状态的模块不同，在输入迅速变化时，求解器不会自动调整时间。当输入是离散信号时，输入的连续微分是一个冲击或者是零。要获得离散信号的离散微分，可以使用下面的公式：

$$y(k) = \frac{1}{\Delta t} (x(k) - x(k-1))$$

或者使用  $z$  传递函数：

$$\frac{Y(z)}{U(z)} = \frac{1 - z^{-1}}{\Delta t} = \frac{z - 1}{z \Delta t}$$

当使用 `linmod` 函数对包含积分模块的系统进行线性化时，会产生困难。

特性

- |                      |     |
|----------------------|-----|
| ➤ Direct Feedthrough | 是   |
| ➤ Sample Time        | 连续  |
| ➤ Scalar Expansion   | N/A |
| ➤ States             | 0   |
| ➤ Vectorized         | 是   |
| ➤ Zero Crossings     | 否   |

## 17. Digital Clock 模块

目的 以定的采样时间间隔输出仿真时间。

库 Sources

说明 Digital Clock 模块只在定的采样时间间隔点输出仿真时间。在别的时间，将保持前面的输出。在离散系统，请使用这个模块来替代 clock 模块。

参数

Sample time

采样时间间隔，缺省值是 1 秒。

特性

- |                    |    |
|--------------------|----|
| ➤ Sample Time      | 离散 |
| ➤ Scalar Expansion | 否  |
| ➤ Vectorized       | 否  |
| ➤ Zero Crossings   | 否  |

## 18. Discrete Filter 模块

目的 实现 IIR 和 FIR 滤波器。

库 Discrete

说明 Discrete Filter 模块实现 IIR 和 FIR 滤波器。用户使用 Numerator 和 Denominator 参数, 按  $z^{-1}$  的升序设置分子多项式和分母多项式的系数。分母的阶数必须大于等于分子的阶数。

数据类型 接收和输出 double 类型的实数信号。

参数

Numerator

定义分子系数的向量, 缺省值是[1]。

Denominator

定义分母系数的向量, 缺省值是[1 2]。

Sample time

在相邻采样间的时间间隔。

特性

- Direct Feedthrough 仅当分子、分母向量的长度相同时才有
- Sample Time 离散
- Scalar Expansion 否
- States 分母参数向量的长度
- Vectorized 否
- Zero Crossings 否

## 19. Discrete Pulse Generator 模块

目的 产生固定间隔的脉冲波形。

库 Sources

说明 这个模块以固定的间隔产生一序列的脉冲。脉冲的周期用脉冲中采样周期的多少来表示。延迟相位可以是正数或者是负数, 但不能比周期大。采样时间只能是正数。这个模块常用在离散或者混合系统, 而在连续系统请用 Pulse Generator 模块。

数据类型 它接受和输出 double 类型的实数信号。

参数

Pulse Generator

脉冲的幅度, 缺省值是 1。

Period

脉冲周期是所占采样点的数目, 缺省值是 2。

Pulse width

一个周期内脉冲处在高电平时所具有的采样点数, 缺省值是 1。

Phase delay

脉冲在产生之前的延迟时间, 用采样点数表示, 缺省值是 0。

Sample time

采样周期，缺省值是 1。

#### 特性

- Sample Time            离散
- Scalar Expansion       参数
- Vectorized             是
- Zero Crossings        否

### 20. Discrete State-Space 模块

目的            实现一个离散状态空间系统。

库              Discrete

说明            DiscreteState-Space 模块实现下式所描述的系统:

$$x(n+1) = Ax(n) + Bu(n)$$

$$y(n) = Cx(n) + Du(n)$$

其中,  $u$  和  $x$  分别代表输入和离散状态,  $y$  是输出。 $A$ 、 $B$ 、 $C$ 、 $D$  是矩阵系数, 必须符合以下规定:

- (1)  $A$  必须是  $n \times n$  的矩阵,  $n$  为状态数;
- (2)  $B$  必须是  $n \times m$  的矩阵,  $m$  是输入信号的宽度;
- (3)  $C$  必须是  $r \times n$  的矩阵,  $r$  是输出的数目;
- (4)  $D$  必须是  $r \times m$  的矩阵。

模块接收一个输入, 产生一个输出, 输入信号的宽度决定于矩阵  $B$  和  $D$  的列数。Simulink 会把包含零的矩阵转换为稀疏矩阵, 实现有效的乘法运算。

数据类型       接收和输出 double 类型的实数信号。

#### 参数

$A, B, C, D$

矩阵系数, 它们的定义见说明。

Initial conditions

初始状态向量, 缺省值是 0。

Sample time

两次采样的时间间隔。

#### 特性

- Direct Feedthrough    只在  $D$  不为零时才是。
- Sample Time            离散
- Scalar Expansion       初始状态
- States                  由矩阵  $A$  的大小决定
- Vectorized             是
- Zero Crossings        否

### 21. Discrete-Time Integrator 模块

目的            实现一个信号离散积分。

库 Discrete  
说明 Discrete-Time Integrator 模块能够在构建纯离散系统时，替代 Integrator 模块。Discrete-Time Integrator 允许用户做的事情有：

- (1) 在模块对话框定义初始条件或者通过输入到模块来定义；
- (2) 输出模块状态；
- (3) 定义积分的上下限；
- (4) 通过一个额外的 reset 输入来重置状态。

#### 积分方法

这个模块采用的方法有：前向欧拉、后向欧拉和 Trapezoidal。对一个给定的时间步  $k$ ，Simulink 更新  $y(k)$  和  $x(k+1)$ 。 $T$  是采样周期，取值根据上限或者下限来翻转。

(1) 前向欧拉方法（缺省），对于这种方法， $1/s$  被近似为  $T/(z-1)$ ，这就得到下面的公式：

$$y(k) = y(k-1) + T * u(k-1)$$

将其转换成状态方程为：

$$\begin{aligned} x(k+1) &= x(k) + T * u(k) \\ y(k) &= x(k) \end{aligned}$$

于是就有：

第 0 步：  $y(0) = x(0) = IC$  在必要时翻转

$$x(1) = x(0) + T * u(0)$$

第 1 步：  $y(1) = x(1)$

$$x(2) = x(1) + T * u(1)$$

第  $k$  步：  $y(k) = x(k)$

$$x(k+1) = x(k) + T * u(k) \quad \text{在必要时翻转}$$

在这种方法下，输入端口 1 没有直接馈入。

(2) 后向积分方法。对于这种方法  $1/s$  被近似为  $T * z/(z-1)$ ，据此得到的状态方程为：

$$\begin{aligned} x(k+1) &= y(k) \\ y(k) &= x(k) + T * u(k) \end{aligned}$$

于是整个算法为：

第 0 步：  $y(0) = x(0) = IC$  在必要时翻转

$$x(1) = y(0)$$

第 1 步：  $y(1) = x(1) + T * u(1)$

$$x(2) = y(1)$$

第  $k$  步：  $y(k) = x(k) + T * u(k)$

$$x(k+1) = y(k)$$

在这种方法，输入端口 1 有直接馈入。

(3) Trapezoidal method。对于这种方法  $1/s$  被近似为  $T * (z+1)/(z-1)$ ，据此得到状态方

程为：

$$y(k) = x(k) + T/2 * u(k)$$

$$x(k+1) = y(k) + T/2 * u(k)$$

这里， $x(k+1)$  是下一个输出的最好近似，它不等于状态，即  $x(k)$  不等于  $y(k)$ 。在这种近似下，算法的流程为：

第 0 步：  $y(0) = x(0) = IC$  在必要时翻转

$$x(1) = y(0) + T/2 * u(0)$$

第 1 步：  $y(1) = x(1) + T/2 * u(1)$

$$x(2) = y(1) + T/2 * u(1)$$

第  $k$  步：  $y(k) = x(k) + T/2 * u(k)$

$$x(k+1) = y(k) + T/2 * u(k)$$

当  $T$  是一个变量时，就有：

$$x(k+1) = y(k)$$

$$y(k) = x(k) + T/2 * (u(k) + u(k+1))$$

据此不难写出此时的积分算法。这种方法下，输入端口 1 有直接馈入。

### 定义初始条件

可以把初始条件作为模块对话框参数来定义，或者从外部信号输入：

(1) 定义初始条件作为模块参数，设定 Initial condition source 参数为 internal，并且在 Initial condition 域输入值；

(2) 从外部源提供初始条件，设定 Initial condition source 参数为 external。一个额外的输入端口将出现在输入端口的下方。

### 使用状态端口

在两种条件下，用户必须使用状态端口替代输出端口：

(1) 当模块的输出通过 reset 端口或者初始条件端口反馈到模块，构成一个代数环时。这种情形的示例，请见 Simulink 的 Bounce 模型。

(2) 当用户想状态从一个条件执行子系统传递到另外一个时，这会产生一个定时问题。这种情形的示例是 Clutch 模型。

可以通过将状态通过状态端口而不是输出端口传递，来校正这些问题。尽管值是相同的，但 Simulink 产生它们的时间略有不同，这就可以使你的模型避免上述问题。可以通过选中 Show state port 来显示状态端口。

缺省情况下，状态端口出现在模块的顶部。

### 为积分设限

为了防止输出超过某个特定水平，可以选中 Limit output 检查框并且在相应的参数域输入限定值。这样的设置就使得模块成为一个限值积分器，当输出超出限定范围，模块将停止积分。在仿真中，用户可以改变限值，但不能改变哪个信号受限。输出按以下的几种情况来确定：

(1) 当积分值小于下饱和限，并且输入是负的，输出就保持在下饱和限；

- (2) 当积分值处于上、下饱和限之间，那么输出就是积分值；
- (3) 当积分值大于上饱和限，而且输入是正值，则输出保持在上饱和限。

产生一个表示处于限值积分状态的信号，可以选择 **Show saturation port** 检查框，这样一个饱和端口会出现在模块输出端口下。它的输出值有三种：

- (1) 1——表示上限正在被应用；
- (2) 0——表示积分值未受限；
- (3) -1——表示下限正在被应用。

当 **Limit output** 选项被选择，模块具有 3 种过零检测：一个是检测积分值何时进入上饱和限，一个是检测积分值何时进入下饱和限，另一个是检测它何时离开饱和。

### 重置状态

模块可以基于一个外部信号将状态重置为设定的初始条件。这时，需选中 **External reset** 选项。一个触发端口将出现在输入端口的下方，触发信号支持的类型有：

- (1) **rising**，当触发信号有上边沿时，重置状态；
- (2) **falling**，当触发信号有下边沿时，重置状态；
- (3) **either**，当触发信号有上边沿或者下边沿时，重置状态。

当模块的输出直接反馈到这个端口时，或者通过一系列具有直接馈入的端口连接到这个端口，重置端口就有直接馈入，这样就有可能产生一个代数环。对这个问题解决的办法是将状态端口接入到重置端口。

**数据类型** 该模块接收和输出 **double** 类型的实数信号。

### 参数

#### Integrator method

积分方法，缺省值是 **ForwardEuler**。

#### External reset

在触发事件发生时重置状态到它们的初始条件。

#### Initial condition source

确定获得状态初始条件的途经是从 **Initial condition** 参数还是外部模块。

#### Initial condition

状态的初始条件，设置 **Initial condition source** 参数为 **internal**。

#### Limit output

如果被选中，限制状态值在 **Lower saturation limi** 和 **Upper saturation limit** 参数之间。

#### Upper saturation limit

积分值的上限，缺省值是 **inf**。

#### Lower saturation limit

积分值的下限，缺省值是 **-inf**。

#### Show saturation port

如果选中，则会增加一个饱和输出端口到模块。

#### Show state port

如果被选中，则为模块增加一个状态输出端口。

**Sample time**

采样时间间隔，缺省值是 1。

**特性**

- **Direct Feedthrough** 是，重置端口和初始条件外部源端口具有
- **Sample Time** 离散
- **Scalar Expansion** 参数
- **states** 从驱动模块的状态和参数继承
- **Vectorized** 是
- **Zero Crossings** 三个，见说明

**22. Discrete Transfer Fcn 模块**

**目的** 实现一个离散传递函数。

**库** Discrete

**说明** Discrete Transfer Fcn 模块实现一个离散  $z$  变换传递函数，它根据下式来定义：

$$H(z) = \frac{num(z)}{den(z)} = \frac{num_0 z^n + num_1 z^{n-1} + \cdots + num_m z^{n-m}}{den_0 z^n + \cdots + den_n}$$

其中， $m+1$  和  $n+1$  分别是分子和分母系数的个数， $num$  和  $den$  包含了分子多项式和分母多项式按  $z$  的降幂排列的系数。 $num$  可以是一个向量或者矩阵， $den$  必须是一个向量，并且都可以在模块参数对话框设定。分母多项式的阶数必须不小于分子多项式的阶数。

模块的输入是一个标量，而输出信号的宽度等于分子多项式的行数。Discrete Transfer Fcn 模块将会在模块图标显示传递函数的分子多项式和分母多项式。

**数据类型** Discrete Transfer Fcn 模块接收和输出 **double** 类型的实数信号。

**参数****Numerator**

设定分子系数的行向量，一个具有多个行的矩阵将认为是设定多个输出，缺省值是[1]。

**Denominator**

设定分母多项式系数的行向量，缺省值是[1 0.5]。

**Sample time**

采样点的时间间隔，缺省值是 1。

**特性**

- **Direct Feedthrough** 仅在分子参数和分母参数的长度相同时才有。
- **Sample Time** 离散
- **Scalar Expansion** 否
- **Vectorized** 否
- **Zero Crossings** 否

## 23. Discrete Zero-Pole 模块

目的	实现一个用零极点形式表示离散传递函数。
库	Discrete
说明	Discrete Zero-Pole 模块实现一个由零极点形式表示离散状态函数。对于一个单输入和单输出系统，它的表示形式为：

$$H(z) = K \frac{Z(z)}{P(z)} = K \frac{(z - Z_1)(z - Z_2) \cdots (z - Z_m)}{(z - P_1)(z - P_2) \cdots (z - P_n)}$$

其中， $Z$  代表零点向量， $P$  代表极点向量， $K$  代表增益。注意极点的数目必须大于或者等于零点的数目。如果零极点是复数向量，那它们必须用复数的坐标对表示。

数据类型 接收和输出 double 类型的实数信号。

参数

Poles

极点的向量，缺省值是[0 0.5]。

Gain

增益，缺省值是 1。

Sample time

采样的时间间隔。

特性

➤ Direct Feedthrough	是，仅当零极点数目相同时。
➤ Sample Time	离散
➤ Scalar Expansion	否
➤ states	极点向量的数目
➤ Vectorized	否
➤ Zero Crossings	否

## 24. Display 模块

目的	显示输入的值。
库	Sinks
说明	显示输入的值，用户可以通过 format 选项来控制显示格式：

- (1) short, 显示 5 位的定点十进制数；
- (2) long, 显示 15 位的定点十进制数；
- (3) short\_e, 显示 5 位的浮点十进制数；
- (4) long\_e, 显示 15 位的浮点十进制数；
- (5) bank, 将输入按元和分来显示值。

使用模块作为一个浮动显示，需选中 Floating display 检查框，模块的输入端口将消失并且在所选择的线显示信号的值。如果选中了 Floating display，那么 Simulink 的缓存复用功能就要关闭。

数据显示的数量和时间取决于模块参数：

- (1) Decimation 参数使模块在每  $n$  个采样显示数据，其中  $n$  是 decimation 因子，其缺省



值是 1。

(2) **Sample time** 参数用于设定模块每次显示数据的时间间隔。这个参数在使用一个可变步长的解法器时非常有用，因为时间步的间隔有可能不相同。缺省值为 -1，表示忽略采样间隔参数。

如果输入是一个向量，模块可以改变大小来显示更多的元素，大小可以在水平和垂直方向改变，改变时，模块在相应的方向增加显示区域。图 8-3 显示了这样的情形。

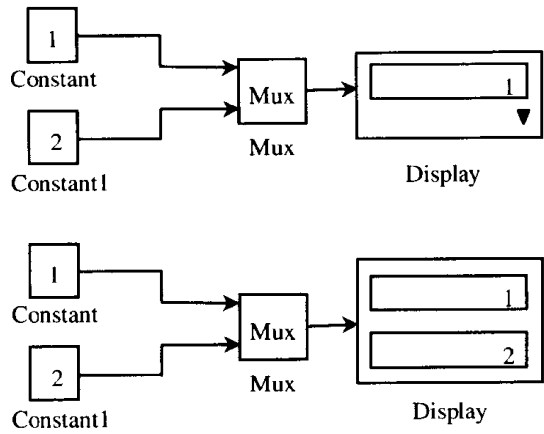


图 8-3 改变模块的显示区域

从图中可以看出，由于 **Display** 模块显示的是一个向量，在用户没有扩大模块的大小时，模块用一个向下的箭头表示还有其他的元素。在调整大小之后，**Display** 模块的显示区域就分成了两个域来分别显示各个元素信号。

**数据类型**      接收和输入任何类型的实数和复数信号。

**参数**

**Format**

数据显示的格式，缺省值是 short。

**Decimation**

显示数据的频率，缺省值是 1，在每个输入点都显示数据。

**Floating display**

如果被选中，模块的输入端口消失，这可以使模块作为一个浮动显示模块。

**Sample time**

显示点的采样时间。

**特性**

- **Sample Time**                      从驱动模块继承。
- **Vectorized**                        是。

**25. Dot Product 模块**

**目的**                      生成点积。

**库**                        Math

**说明**                      **Dot Product** 模块生成两个输入信号的点积，输出值是一个标量信号。其对

应的操作为

$$y = u1' * u2$$

如果两个输入都是向量，那么它们必须长度相同。

特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承。
- Scalar Expansion 是
- states 0
- Vectorized 是
- Zero Crossings 否

26. First-Order Hold 模块

目的 实现一阶采样和保持。

库 Discrete

说明 First-Order Hold 模块实现一个按设定采样间隔进行的一阶采样保持。这个模块在实际应用中作用不大，仅仅是作为教学用。运行演示模型 fohdemo 可以看出零阶保持和一阶保持模块的区别。图 8-4 显示了一个正弦波和它经过一阶保持模块后的输出。

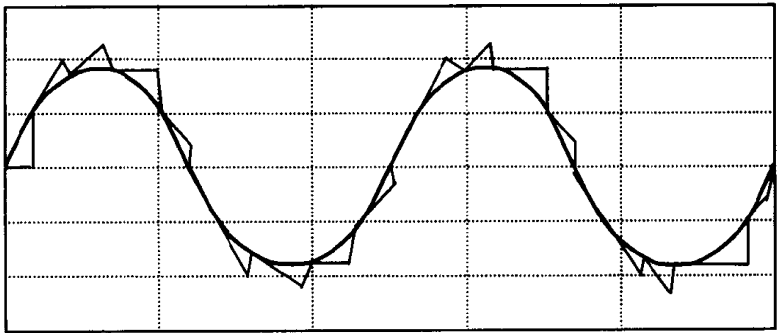


图 8-4 正弦波的一阶保持输出

数据类型 接收和输出 double 类型的信号。

参数

Sample time  
采样的时间间隔。

特性

- Direct Feedthrough 否
- Sample Time 连续
- Scalar Expansion 否
- states 对应每个输入元素有一个连续和一个离散状态。
- Vectorized 是
- Zero Crossings 否

27. From 模块

目的 从 Goto 模块接收输入。

库 Signals&Systems

说明 From 模块接收来自相应 Goto 模块的信号，并把它作为输出。输出的数据类型和来自 Goto 模块的输入相同。From 模块和 Goto 模块允许将信号无需真实连接的从一个模块传到另一个模块（图 8-5）。将一个 Goto 模块和一个 From 模块关联，可以在 From 模块的 Goto tag 参数输入 Goto 模块的标签。

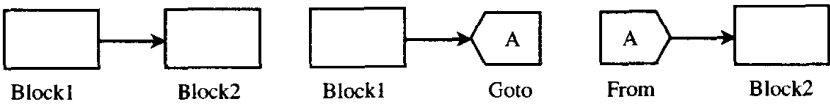


图 8-5 From 模块作用示意图

一个 From 模块仅能从一个 Goto 模块接收信号，尽管一个 Goto 模块可以把它的信号传给多个 From 模块。

相互关联的 Goto 和 From 模块可以出现在模型的任何位置，但有一个例外：如果其中一个在一个条件子系统中，那么另一个模块必须在相同的子系统或者该子系统所包含的子系统（条件执行子系统除外）。然而，如果一个 Goto 模块连接到一个状态端口，这个信号就可以被送到另一个条件执行子系统中。

Goto 模块标签的作用域决定了可以接受数据的 From 模块，Goto 模块的图标能反映出标签的作用域：

- (1) 一个局部标签名用中括号括起来；
- (2) 一个范围标签名用大括号括起来；
- (3) 全局标签名则没有附加字符。

数据类型 接受任何类型的实数或者复数信号。

参数

Goto tag 传递信号到 From 模块的 Goto 模块的标签。

特性

- Sample Time 从驱动的 Goto 模块继承
- Vectorized 是

28. From File 模块

目的 从一个文件读数据。

库 Sources

说明 From File 模块从设定的文件读取数据。模块图标显示提供数据的文件名。文件内必须包含有两行或者更多行的矩阵。第一行必须包含一个单调增加的时间点，其他的行包含了所在列的时间点所对应的数据。

输出的宽度取决于文件中矩阵的行数。模块根据时间数据来决定每个时间点的输出，但

是不输出时间值。这意味着对于一个  $m$  行的矩阵，模块输出一个长度为  $m-1$  的向量。如果要在文件设定的两个相邻时间数据之间的时间点输出数据，那么将在相应值间通过线性内插来获得数据。如果要求的时间点小于第一个时间值或者大于最后一个时间值，Simulink 会使用前面两个值或者后面两个值进行外插得到。如果在相同的时间值有两列或者更多列，那么 Simulink 取首先遇到的第一列的数据输出。例如，对于一个有下面数据的矩阵：

时间值：0 1 2 2

数据点：2 3 4 5

在时间点 2，输出 4，而不是 5。

因为 Simulink 在仿真开始时将数据读到内存，因此，不能从相同模型 To File 模块写数据的文件读取数据。

数据类型      double 类型的实数数据。

参数

File name

包含作为输入的数据的文件，缺省文件名是 untitled.mat。

Sample time

从文件读取数据的采样率。

特性

- Sample Time                      从驱动模块继承
- Scalar Expansion                否
- Vectorized                        是
- Zero Crossings                  否

29. From Workspace 模块

目的              从工作空间读取数据。

库                Sources

说明              From Workspace 模块从 MATLAB 工作空间读取数据。模块的 Data 参数通过一个 MATLAB 表达式，对包含信号数据和时间步的矩阵或者结构进行估值，以此设定工作空间的数据。矩阵或者结构的格式和用于从工作空间载入输入数据的相同。From Workspace 模块显示 Data 参数里的表达式。

如果输入表格没有指定输入数据的时间，那么每个值就被假定时间  $t = (n-1) * st$ ，其中， $n$  是第  $n$  个输入值， $st$  是模块的采样时间。

From Workspace 模块的在每一个仿真时间步的输出取决于模块的 Interpolate data 和 Hold final data value 参数的设置。表 8-1 总结了在参数设置的不同组合下的模块行为。

表 8-1                      不同组合的参数设置下的模块行为

插 值 选 项	保 持 选 项	模块输出 ( $t_i < t < t_f$ )	模块输出 ( $t > t_f$ )
on	off	在数据值间内插	从最后的数据值外插
on	on	在数据值间内插	最后的数据值
off	off	取最接近的数据值	0
off	on	取最接近的数据值	最后的数据值

如果对于相同的时间值，有多个数据值和它对应，那么 Simulink 会使用最后一个设定的数据值。例如，

时间：0 1 2 2

数据：2 3 4 5

在时间 2，输出的值是 5。

注意，From Workspace 模块可以直接读取 To Workspace 模块的输出，如果输出是结构或者带时间的结构格式，当输出的是矩阵格式时，需要增加一个时间值。

数据类型 能够输出任何数据类型的实数或者复数值。

参数

#### Data

计算（产生）包含仿真时间和相应信号值的结构或者矩阵。例如假定工作空间包含一个名为 T 的列向量，并且包含相应信号值的矩阵是 U。那么，缺省的表达式就是[T,U]，如果所需的信号时间矩阵或者结构已经存在于工作空间，简单的输入对应的变量名就可以了。

#### Sample time

来自工作空间的数据的采样速率。

#### Interpolate data

这个选项使模块在工作空间的数据变量没有已知的点和它对应时，采用线性内插来获得输出。否则就选取已知点中在时间上与之最接近的点作为输出。

#### Hold final data value

这个选项使模块将保持上一个输出值，至此当前值出现。

特性

- Sample Time 从驱动模块继承
- Scalar Expansion 否
- Vectorized 是
- Zero Crossings 否

### 30. Function-Call Generator 模块

目的 以设定的速率执行 Function-Call 调用子系统。

库 Signals&Systems

说明 Function-Call Generator 模块按模块 sample time 参数指定的采样速率执行函数调用子系统。实现按预先规定的顺序执行多个 Function-Call 子系统，可以把 Function-Call Generator 连接到 demux 模块的输入端口，demux 端口的输出和要控制的 Function-Call 子系统数目相同，然后再 demux 的输出和要控制的子系统连接起来。连接到 demux 第一个端口的系统首先执行，连接到第二个的将第二个执行，依此类推。

数据类型 输出 double 类型的实数信号。

参数

Sample time

采样点的时间间隔。

特性

- Direct Feedthrough 否
- Sample Time 用户设定
- Scalar Expansion 否
- Vectorized 是
- Zero Crossings 否

### 31. Gain 模块

目的 将模块输入乘上一个增益。

库 Math

说明 Gain 模块将它的输入乘上一个设定的常数、变量或表达式，来获得输出。用户可以在 gain 编辑框输入数值，或者一个变量，或者一个表达式。将输入乘上一个矩阵，使用 Matrix Gain 模块。Gain 模块图标会显示在 Gain 参数域输入的值，只要它的图表足够大。如果 gain 被设定为一个变量，模块显示变量名，但是如果变量是在原括号里定义的，模块在每次被重画时会对变量进行估值，并且在图标上显示值。如果 Gain 参数的值太长，字符串 -K- 被显示。

在仿真运行时改变增益，可以使用 Slider Gain 模块。

数据类型 Gain 模块接收除布尔类型的任何类型的实数或者复数信号，并且输出和输入相同的数据类型。

参数

Gain

增益可以被设定为标量、向量、变量名或者表达式。缺省值是 1。如果没有被指定，Gain 参数的数据类型是 double。

Saturate on integer overflow

如果被选中，选项使 Gain 模块的输出在整数溢出时饱和。特别的，如果输出的数据类型是整数类型，模块输出就是输出类型对计算的输出所能表示的最大值。如果选项没有被选中，那么 Simulink 按照仿真参数对话框的诊断页的设定的行为来进行。

特性

- Direct Feedthrough 是
- Sample Time 从驱动模块集成
- Scalar Expansion 输入和增益参数
- states 0
- Vectorized 是
- Zero Crossings 否

### 32. Goto 模块

目的 将模块输入传到 From 模块。

库 Signals&Systems

说明

Goto 模块将它的输入传给他对应的 From 模块。输入可以任何类型的实数或者复数信号。通过 Goto 模块和 From 模块可以不用连接就可以传递信号。一个 Goto 模块可以把信号传给多个 From 模块。Goto 模块和 From 模块是通过标签来匹配的。标签具有作用域（可视域），它决定能够访问 Goto 模块的信号的 From 模块的位置：

（1）local，缺省情况，表示使用相同标签的 From 和 Goto 模块必须在相同的子系统。一个局部标签用中括号包含。

（2）scoped，表示使用相同标签的 From 和 Goto 模块必须在相同的子系统，或者是在 Goto 标签可视模块所在子系统的下层子系统。这种标签用大括号包含。

（3）global，表示使用相同标签的 From 和 Goto 模块可以在模型的任何位置。

当使用相同的 tag 标签的 Goto 和 From 模块处在相同的子系统时，就要使用 local 标签。如果这两个模块不处在相同的子系统，就必须使用 global 或者 scoped 标签。当用户定义一个标签为 global，对那个标签的所有调用都访问相同的信号。定义为 scoped 的标签可以在模型的任何位置使用。图 8-6 显示了一个 scoped 标签的使用示例。

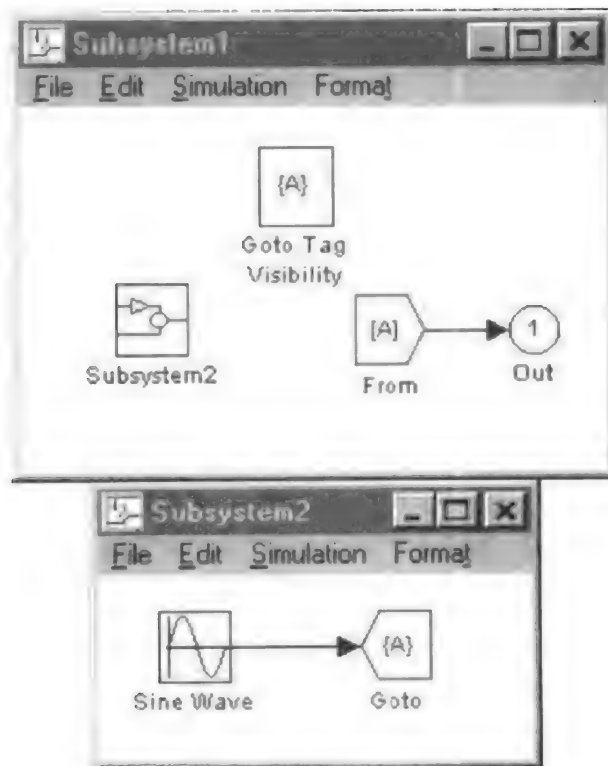


图 8-6 scoped 标签使用示例

数据类型

接收和输出任何类型的实数或者复数信号。

参数

Tag

Goto block 标识，这个参数标识了 Goto block。

Tag visibility

Goto block 的作用域：local、scoped 和 global。缺省是 local。

特性

- Sample Time                  采样时间
- Vectorized                    是

33. Ground 模块

目的                  将未连接端口接地。

库                    Signals&Systems

说明                  Ground 模块可以用来连接具有未连接端口的输入模块。如果用户运行一个存在未连接输入端的模块的仿真，Simulink 会给出一个警告信息。使用 Ground 模块可以避免这些警告。Ground 模块输出一个零值信号，信号的数据类型和所连接端口的类型相同。

数据类型            Ground 模块输出的信号具有和所连接端口的数据类型相同的数值类型（实数或者复数），请看图 8-7 所示的示例模型。模型中，Ground 模块输出信号的数据类型和加法器的类型相同，而加法器的类型受 Constant 模块的控制，所以 Ground 模块输出 int8 类型的数据。

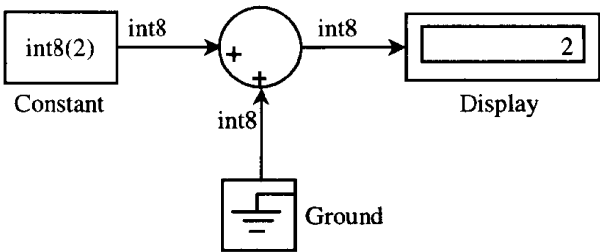


图 8-7 Ground 模块示例模型

特性

- Sample Time                  从驱动模块接触
- Vectorized                    是

34. Hit Crossing 模块

目的                  检测过零点。

库                    Signals&Systems

说明                  Hit Crossing 模块检测何时输入按 Hit crossing direction 所指定的方向达到 Hit crossing offset 参数值。模块在机器精度范围内找到过零点。Show output port 选项被选择，模块的输出表示何时过零点到达。如果输入信号恰巧是 offset 的值，模块在当前仿真步输出 1。如果输入信号在偏移值的两个临近点区域中，模块在第二个时间步输出 1。Show output port 检查框没有被选中，模块只能仿真但不产生输出。

Hit Crossing 模块在某些数值和计算机精度的限制下，相当于“几乎相等”模块。Simulink 的演示模型 hardstop 和 clutch 演示了 Hit Crossing 模块的使用。

数据类型            Hit Crossing 模块输出一个布尔类型的信号，但如果布尔兼容模式被启用，模块数一个 double 类型的信号。



参数

- Hit crossing offset  
需要被检测到达的值。
- Hit crossing direction  
输入信号到达 Hit crossing offset，产生一个过零点方向。
- Show output port  
如果被选中，添加一个输出端口。

特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承。
- Scalar Expansion 是
- Vectorized 是
- Zero Crossings 是，检测过零点。

35. IC 模块

- 目的 设置信号的初始值。
- 库 Signals&Systems
- 说明 IC 模块设置了连接到它的输出端口的信号的初始值。具体讲，该模块在  $t=0$  时刻，将信号的值设为模块参数设置的值，然后再根据输入的值来确定信号的值。这个模块在为代数环提供初始猜测时非常有用。图 8-8 演示了如何使用 IC 模块设置信号的初始值，示例中，被初始化的信号是 test signal。

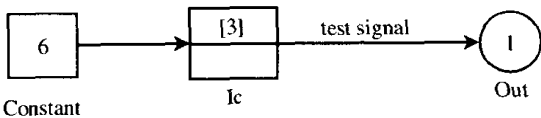


图 8-8 用 IC 模块初始化信号

- 数据类型 接收和输出 double 类型的实数和复数信号。
- 参数

Initial value  
信号的初始值，缺省值是 1。

特性

- Direct Feedthrough 是
- Sample Time 从参数继承
- Scalar Expansion 仅对参数
- states 0
- Vectorized 是
- Zero Crossings 否

36. look-Up Table 模块

- 目的 实现对输入的分段线性匹配。

**库** Functions&Tables

**说明** Look-Up Table 模块使用对在模块参数定义的值, 使用线性内插将输入匹配到输出。用户可以通过设定 **Vector of input values** 和 **Vector of output values** 参数设定向量来定义表格。模块通过将模块输入和输入向量的值比较来产生输出值。

(1) 如果找到一个值和模块的输入匹配, 输出是输出向量的对应元素;

(2) 如果没有找到匹配的值, 它对表格中两个接近的值进行线性内插来获得输出值, 如果模块输入小于第一个或者是大于最后一个输入向量元素, 模块就用最前面的两个或者最后面的两个元素来外插获得。

将两个输入匹配到输出, 使用 Look-Up Table (2-D) 模块。建立一个跳变, 对不同的输入值重复相同的输出值。例如, 下面定义的表格

输入值向量: [-2 -1 -1 0 0 1 1 2]

输入值向量: [-1 -1 -2 -2 1 2 2 1 1]

图 8-9 是这个表格画出的图。

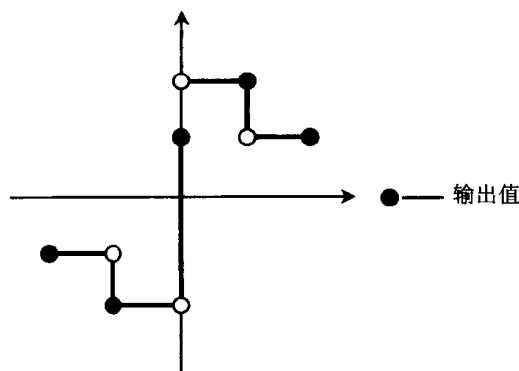


图 8-9 look-Up Table 模块的示例

**数据类型** 接收和输出 **double** 类型的信号。

**参数**

**Vector of input values**

包含可能的模块输入值的向量, 这个向量必须和输出向量的大小相同, 输入向量必须单调增加。

**Vector of output values**

包含模块输出值的向量, 向量必须和输入向量的大小相同。

**特性**

- **Direct Feedthrough** 是
- **Sample Time** 从驱动模块继承。
- **Scalar Expansion** 否
- **Vectorized** 是
- **Zero Crossings** 否

## 37. Memory 模块

目的	输出前一个仿真时间步的模块输入。	
库	Continuous	
说明	Memory 模块输出前一个时间步的输入。	
数据类型	Memory 模块接收任何数值类型的信号，包括用户定义的类型。如果输入类型是用户定义的，初始条件是 0。	
参数	<p><b>Initial condition</b> 在初始积分步的输出。</p> <p><b>Inherit sample time</b> 选中这个检查框使采样时间从驱动模块继承来。</p>	
特性	<ul style="list-style-type: none"> <li>➤ Direct Feedthrough      否</li> <li>➤ Sample Time              连续，但在 Inherit sample time 检查框选中时，从驱动模块继承。</li> <li>➤ Scalar Expansion        Initial condition 参数。</li> <li>➤ Vectorized                是</li> <li>➤ Zero Crossings          否</li> </ul>	

## 38. Merge 模块

目的	将输入线连接成一个标量信号。
库	Signals&Systems
说明	Merge 模块将它的输入连接成单个的输出线，它在任何时间的值等于它的驱动模块的最近计算的输出。

用户通过设置 Number of Inputs 参数来设定输入的个数。所有的输入必须具有相同的宽度。可以设置 Initial Output 参数来设定初始输出。如果没有指定输出，而一个或者多个驱动模块设定了，那么 Merge 模块从驱动模块中继承最近估值的初始输出。Merge 模块便于产生交替执行的子系统。Simulink 限制可以加到 Merge 模块的连接类型。特别的，它只允许建立从非虚拟模块到 Merge 模块的输入的一对一匹配的连接。Simulink 不允许将多个非虚拟输出连接到一个 Merge 模块的单个输入上。

Merge 模块不接受元素被重排过的信号，图 8-10 演示了这个情形。在图中，由于 Selector 将输入信号的元素重排，所以 Merge 模块不接受它的输出。

数据类型	接收任何数值类型的信号，包括用户自定义类型，如果是自定义类型，初始条件必须是 0。
------	---

参数	<p><b>Number of inputs</b> 输入端口的个数。</p> <p><b>Initial output</b> 输出的初始值。如果没设定，初始输出等于驱动模块的初始模块值。</p>
----	---

特性

- Sample Time 从驱动模块继承。
- Scalar Expansion 否
- Vectorized 是

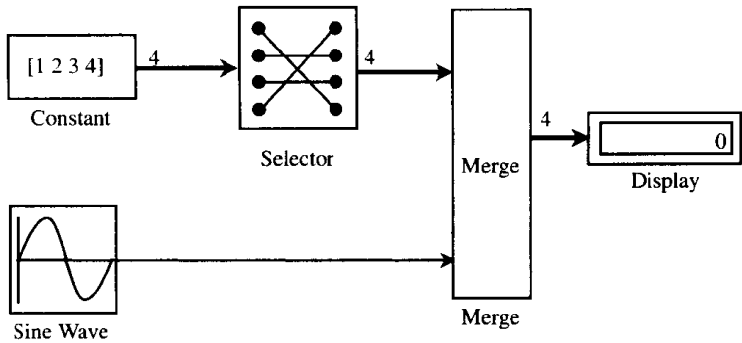


图 8-10 Merge 模块不接受元素被重排的信号示例

39. Probe 模块

**目的** 探查一条直线的宽度，采样时间或者复数信号标记。

**库** Signals&Systems

**说明** Probe 模块输出它的输入信号的有关信息，显示信息的内容可以由用户在模块参数设置。该模块可以输出信号的宽度，采样时间以及复数信号标记，所谓复数信号标记表明该信号是否为复数信号。模块输出端口的个数取决于所选择的要探查的信息数目，它们是信号宽度，采样时间和复数信号标记。每一种信息的值在一个单独的输出端作为独立的信号输出。模块接受实数或者复数信号，也可以是任何内置数据类型的向量。在仿真过程中，模块的图标显示探查到的数据。

**数据类型** 见说明部分。

参数

Probe width

这个检查框如果被选中，表明要探查信号的宽度。

Probe sample time

这个检查框如果被选中，表明要探查信号的采样时间。

Probe complex signal

这个检查框如果被选中，当输入信号是复数信号，模块输出 1，不是，则输出 0。

特性

- Direct Feedthrough 否
- Sample Time 从驱动模块继承
- Scalar Expansion 是
- Vectorized 是

	Zero Crossings	否
40. Quantizer 模块		
目的	按一个设定的间隔将输入离散化。	
库	Nonlinear	
说明	Quantizer 模块将输入传给一个接替函数，将输入轴的许多临近点匹配到输出轴上的一个点。它的效果是将一个连续信号量化为一个条形信号。输出和输入的关系为：	

$$y = q * \text{round}(u / q)$$

其中， $y$  是输出， $u$  是输入， $q$  是量化间隔参数。

数据类型 接受和输出 double 类型的信号。

参数

Quantization interval  
输出被量化的间隔，缺省值是 0.5。

特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承。
- Scalar Expansion 参数可以
- Vectorized 是
- Zero Crossings 否

#### 41. Ramp 模块

目的	产生一个常速率增加或者减小的信号。
库	Sources
说明	产生一个起始于设定时间和设定值，并按设定值变化的信号。
数据类型	输出 double 类型的信号。
参数	

Slope  
产生信号的变化速率，缺省值是 1。

Start time  
信号开始被产生的时间，缺省值是 0。

Initial output  
信号的初始值，缺省值是 0。

特性

- Sample Time 从驱动模块继承
- Scalar Expansion 是
- Vectorized 是
- Zero Crossings 是

#### 42. Relay 模块

目的	在两个常数间切换输出。
----	-------------

库 Nonlinear

说明 Relay 模块允许输出在两个设定值间切换, 当 Relay 开启时, 它会一直保持开启直至输入降到 Switch off point 参数的值。当 relay 关闭时, 它会保持关闭, 直至输入超过 Switch on point 参数的值。

Switch on point 的值应该大于 Switch off point 的值。

数据类型 接受和输出 double 类型的实数信号。

参数

Switch on point

接续的开启门限, 缺省值是 eps。

Switch off point

接续的关闭门限, 缺省值是 eps。

Output when on

当接续开启时的输出, 缺省值是 1。

Output when off

当接续关闭时的输出, 缺省值是 0。

特性

- Direct Feedthrough 是
- Sample Time 从驱动模块继承
- Scalar Expansion 是
- Vectorized 是
- Zero Crossings 是, 检测开关开启和关闭点

#### 43. Selector 模块

目的 选择输入元素。

库 Signals&Systems

说明 Selector 模块有选择的输出输入向量的元素。Elements 参数定义了输入向量在输出向量的次序。这个参数必须设定为向量, 除非只有一个元素被选择。模块显示输入元素的输出顺序。

数据类型 接收和输出任何类型的信号。

参数

Elements

输入信号的元素出现在输出向量的持续。

Input port width

输入向量元素的数目。

特性

- Sample Time 从驱动模块继承
- Vectorized 是

#### 44. Slider Gain 模块

目的 使用一个滑动器改变标量增益。

库 Math

说明	Slider Gain 模块一个滑动器让用户在仿真时改变增益。这个模块接受一个输入和产生一个输出。	
数据类型	同 Gain 模块	
参数	<p>Low 滑动范围的下限，缺省值是 0。</p> <p>High 滑动范围的上限，缺省值是 2。 The upper limit of the slider range. The default is 2. 对话框上的编辑域（从左到右）表示下限、当前值和上限。改变增益的方法有两种：操纵滑动条或者在当前值输入一个新值。 当在滑动器的左或者右边的箭头单击，当前值会变化滑动范围的 1%。</p>	
特性	<ul style="list-style-type: none"> <li>➤ Direct Feedthrough      是</li> <li>➤ Sample Time              从驱动模块继承</li> <li>➤ Scalar Expansion        增益</li> <li>➤ states                    0</li> <li>➤ Vectorized                是</li> <li>➤ Zero Crossings          否</li> </ul>	

#### 45. Transport Delay 模块

目的      将输入延迟给定的时间。

库      Continuous

说明      Transport Delay 模块将输入延迟给定的时间。它可以用于模拟时延。在仿真开始时，模块输出 Initial input 参数直至仿真时间超过了 Time delay 参数，这时模块开始产生延迟的输入信号。Time delay 必须是非负数，这是为了保持因果性。

仿真中，模块保存输入点和仿真时间到缓存中，缓存的初始大小由 Initial buffer size 参数来定义。如果数据点的数目超出了缓存空间，模块分配额外的存储空间并且在仿真结束后显示一个信息来表示多少所需的总的缓存。因为分配缓存会减慢仿真，仔细的定义参数值，可以提高仿真的速度。对于长时间的延迟，这个模块需要大量的空间，尤其是对于向量化输入。

当所要求的输出在保存的输入值找不到对应的时间点时，就会采用线性内插技术。当延迟小于仿真步长，就会从上一个仿真点来外插获得数据。因为模块没有直接馈入，它不能使用当前的输入来计算它的输出值。为了说明这一点，考虑一个固定步长为 1 的仿真，并且当前时间在  $t=5$ 。如果延迟是 0.5，模块需要输出保存的  $t=4.5$  的数据。但是最近保存的时间点是 4，所以模块采用前向外插。

数据类型      接收和输出 double 类型的实数信号。

参数

Time delay

输入信号到被传递给输出之间的延迟的仿真时间，这个值只能是非负数。

#### Initial input

在仿真开始和延迟时间之间输出的值。

#### Initial buffer size

为保存的数据点分配的初始内存。

#### 特性

- Direct Feedthrough 否
- Sample Time 连续
- Scalar Expansion 输入和输出，除了初始缓存大小
- Vectorized 是
- Zero Crossings 否

### 46. Zero-Order Hold 模块

目的 实现一个周期的零阶保持。

库 Discrete

说明 Zero-Order Hold 实现按设定的采样速率运行的采样保持功能。模块接收一个输入并且产生一个输出，它们都可以是标量或者是向量。模块提供了一个对一个或者多个信号按不同的速率重新采样的机制。例如，可以把它和 Quantizer 模块结合起来模拟一个输入为连续信号的 A/D 转换器。

数据类型 接收和输出任何数据类型的实数或者复数信号。

#### 参数

#### Sample time

采样的时间间隔，缺省值是 1。

#### 特性

- Direct Feedthrough 是
- Sample Time 离散
- Scalar Expansion 是
- states 0
- Vectorized 是
- Zero Crossings 否





## 第九章 用 S-函数扩展 Simulink

通过前面几章的学习,读者对用 Simulink 建模的基本思路应该有了一个比较清晰的认识, Simulink 为用户提供了许许多多的内置库模块,正如 MATLAB 语言的内置函数一样,用户的主要工作就是将所要解决的问题映射到由这些模块拼搭在一起的系统。为了方便用户更有条理、便捷的管理模型, Simulink 提供了许多的虚拟模块来建立模型。对于那些经常使用的模块组合(子系统), Simulink 允许用户把它们建成子系统,并封装成能够重复使用的库模块,这样就减少许多不必要的重复劳动。可以说,这种封装方式也是一种对 Simulink 进行扩展的方式,但它是基于 Simulink 原来提供的内置模块的。在这一章里,读者会学习到另外一种扩展 Simulink 的强大机制——S-函数,它是 System function——系统函数的简称。

### 9.1 S-函数综述

#### 9.1.1 什么是 S-函数

S-函数是一个动态系统的计算机语言描述,在 MATLAB 里,用户可以选择用 MATLAB 语言还是 C 语言来编写 S-函数, MATLAB6.0 还支持使用 C++ 来写 S-函数。前者在本书里,不妨称为 M 文件 S-函数,而后者则取名为 C-MEX 文件 S-函数。对于后者,读者也许会很陌生,这里姑且放在一边,读者只要记住 C-MEX 文件是 C 语言写的 S-函数在 MATLAB 被编译成的一种可执行文件就行了。

S-函数采用一种特殊的调用语法,使函数和 Simulink 方程解法器进行交互,这种交互与在解法器和 Simulink 间发生的交互十分类似。S-函数的形式十分通用,它能够支持连续系统、离散系统和混合系统,可以说几乎所有的 Simulink 模型都可以用 S-函数来描述。

读者请先看一个简单的 S-函数例程。

这个文件读者无需手动输入,只要用 MATLAB Edit 窗口打开你机器的 MATLAB 安装目录下的 toolbox\ Simulink\ toolbox 目录里的 timestwo 文件即可。它是一个把输入信号乘以 2 倍的模块的 MATLAB 语言描述。而这用 Simulink 的内置模块实现,只需要一个 gain 模块就行了。

```
function [sys,x0,str,ts] = timestwo (t,x,u,flag)
%TIMESTWO S-function whose output is two times its input.
% This M-file illustrates how to construct an M-file S-function that
% computes an output value based upon its input. The output of this
% S-function is two times the input value:
```

```
%      y = 2 * u;
%      See sfuntmpl.m for a general S-function template.
%      See also SFUNTMPL.
%      Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
%      $Revision: 1.3 $

switch flag,
    case 0
        [sys,x0,str,ts]=mdlInitializeSizes;
    case 3
        sys=mdlOutputs (t,x,u) ;
    case { 1, 2, 4, 9 }
        sys=[];
    otherwise
        error (['Unhandled flag = ',num2str (flag) ] ) ;
end
function [sys,x0,str,ts] = mdlInitializeSizes ( )
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = -1; % dynamically sized
sizes.NumInputs      = -1; % dynamically sized
sizes.DirFeedthrough = 1;  % has direct feedthrough
sizes.NumSampleTimes = 1;
sys = simsizes ( sizes ) ;
str = [];
x0  = [];
ts  = [-1 0]; % inherited sample time
% end mdlInitializeSizes
function sys = mdlOutputs (t,x,u)
sys = u * 2;
% end mdlOutputs
```

请读者先把编写 S-函数的种种疑问，搁置在一边，就假定 S-函数已经编好了，先来看看 S-函数是如何被合并到 Simulink 模型里的。这也就是如何在 Simulink 里调用 S-函数的问题。为此，就需要使用 functions&tables 子库里的 S-function 模块。

如图 9-1 所示，请读者在 S-函数的对话框的 S-function name 编辑框内输入要调用的函数名，这里是 timestwo，至于 parameters 参数在这个例子里就不用填写。设置完参数的 S-function 模块就如图 9-1 上的左图显示的一样。下面就可以使用这个模块了，读者不妨建立一个简单

的模型来测试一下它的作用，例如将 `sinewave` 模块作为 S-函数模块的输入，而把它的输出接到一个 `scope` 模块显示。

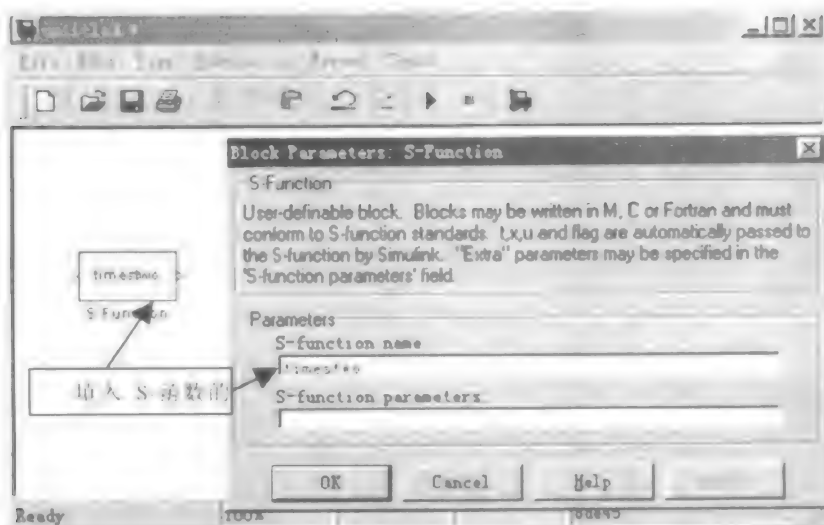


图 9-1 S-函数模块

但上面的模块在被双击后出现的对话框仍然是 S-函数的对话框，而且有时候用户编写的 S-函数还可能传递参数进去，这时再使用这个对话框就不太方便了。比较好的方法，是对 S-function 模块进行封装，自定义 S-函数的参数对话框，使封装后的 S-函数模块表现得与内置模块一样。

这里就有一个问题，什么样的参数才是额外参数呢？

在前面的 S-函数的源代码里，S-函数的说明语句为

```
function [sys,x0,str,ts] = timestwo (t,x,u,flag)
```

上面的这些输入参数 `t`、`x`、`u` 和 `flag` 是编写 S-函数所必须的输入参量，这些参数在设置 S-function 模块参数时不再填写在额外参数编辑框里，也就是说它们由 Simulink 解法器来自动传给 S-函数。除它们之外的任何参数都是额外参数，需要用户从外部传值给它们。因此，额外参数必须按照函数声明的顺序在 S-函数参数对话框的 `parameter` 编辑框里进行说明。例如，要定义一个具有额外参量 `a` 和 `b` 的 S-函数 `sexample`，就必须如图 9-2 所示去设置模块的参数编辑框。

```
function [sys,x0,str,ts] = sexample (t, x, u, flag, a, b)
```

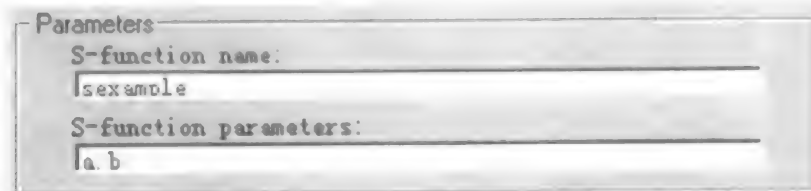


图 9-2 额外参数的设置

读者可以看到，在 `sexample` 函数里 `a, b` 就是额外参数，它们使用时需要用户定义，所以

要在编辑框里说明，而且说明的顺序就是在函数文件里说明的顺序。然后再运用与普通的子系统封装一样的方法来对 S-函数模块进行封装，自定义它的模块图表和参数设置对话框。

前面说过，S-函数最广泛的用途是定制用户自己的 Simulink 模块，更具体地讲，它的作用体现在以下几点：

- (1) 用户可以用它来创建新的通用性的 Simulink 模块；
- (2) 将已存在的 C 代码合并到仿真中；
- (3) 将一个系统描述成一个数学方程；
- (4) 便于使用图形仿真。

使用 S-函数的一个好处就是，用户可以建立能多次使用的通用模块，而模块的每个实例可以具有不同的参数值。

### 9.1.2 S-函数如何工作

在具体讲述 S-函数的工作原理之前，还是先来回顾一下前面讲过的 Simulink 模块的工作模型。

Simulink 中的每一个模块都有三个基本元素：输入向量、状态向量和输出向量，本书分别用  $u$ 、 $x$  和  $y$  来标记它们。图 9-3 反映了三个元素之间的关系。

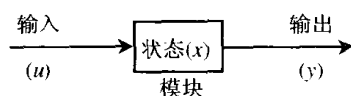


图 9-3 Simulink 模块的基本模型

在上图的三个基本元素中，状态向量无疑是最重要的，也是最灵活的一个概念。何谓状态呢？读者首先想到的可能是线性系统理论里的状态方程里的状态，在那里，状态被定义成一些微分量。可以说那里的状态只是状态这个概念内涵的一部分。不仅仅只是工程系统才存在状态，事实上，状态这个概念在非工程系统也能找到它的对应物。

在 Simulink 里状态向量可以分为连续状态、离散状态，或者是两者的结合。输入、输出、状态这三个量的关系可以用下面的方程来反映。

$$y = f_o(t, u, x)$$

$$x_{d_{k+1}} = f_u(t, u, x)$$

$$x'_c = f_d(t, u, x)$$

$$\text{其中, } x = \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}$$

Simulink 在仿真时把上面的这些方程对应为不同的仿真阶段，它们分别是计算模块的输出、更新它的离散状态和计算连续状态的微分。在仿真的开始和结束，还包括初始化和结束任务两个阶段。在每一个阶段，Simulink 都重复地对模块进行调用。

关于仿真，读者已经接触到了好几个概念，仿真步(simulation step)、仿真阶段(simulation

stage)。这里还要讲一个概念：仿真循环（simulation loop）。一个仿真循环就是由仿真阶段按一定顺序组成的执行序列。对于每个模块，经过一次仿真循环就是一个仿真步，而在同一个仿真时间步，模型中各模块的仿真步按照事先排列好的次序依次执行。这个过程可以用图 9-4 来表示。

从图中可以看出，在仿真开始时，Simulink 首先对模型进行初始化，初始化这个阶段不属于仿真循环。在所有的模块都初始化后，模块才进入仿真循环，在仿真循环的每个阶段，Simulink 都要调用模块或者 S-函数。由于在积分时，对仿真的步长有要求，所以这时需要把仿真的时间步细化。完成一个仿真循环后，就进入下一个仿真步，如此循环直至仿真的结束。在结束时，用户还可以指定仿真来完成某些特殊操作。对于变采样时间模块，Simulink 还要在每个仿真循环开始时，确定下一个仿真步的采样时刻。

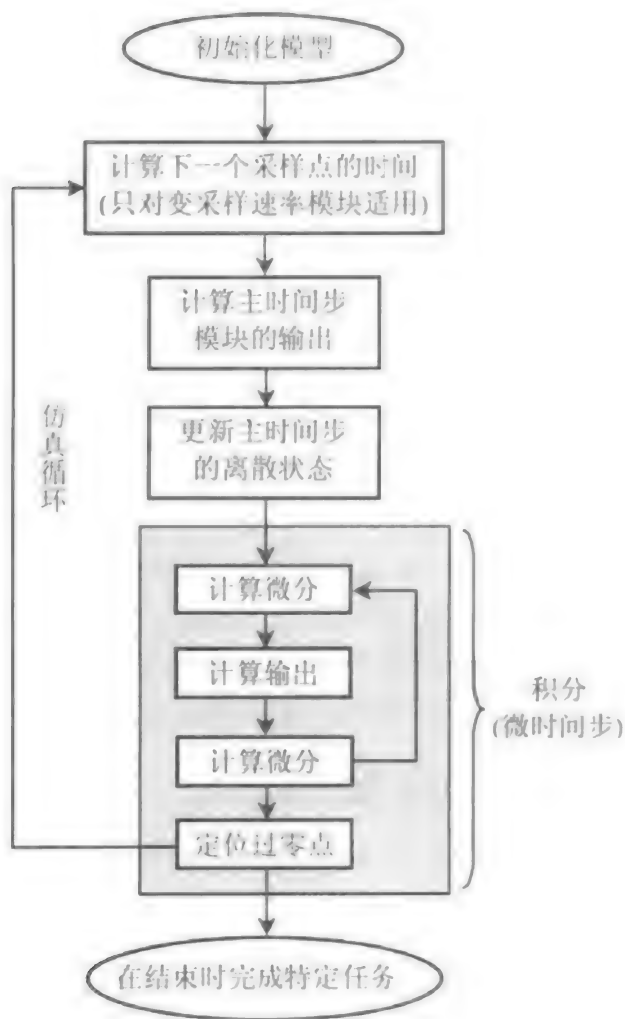


图 9-4 仿真执行流程图

在调用模型中的 S-函数时，Simulink 会调用用户定义的 S-函数方法(S-function routines)，来实现每个仿真阶段要完成的任务。这些任务包括：

- (1) 初始化阶段。它是仿真进行的第一步，先于第一个仿真循环，它初始化 S-函数，

在这个阶段，Simulink 完成下面的工作：

- 初始化 SimStruct——包含 S-函数信息的数据结构；
- 确定输入、输出端口的数目和大小；
- 确定模块的采样时间；
- 分配内存和 sizes 数组。

(2) 计算下一个采样点的时间。如果模型使用变步长解法器，那么就需要在当前仿真步确定下一个采样点的时刻，也即下一个仿真步的大小。

(3) 计算当前主仿真步的输出。在这个调用完成之后，模块的所有输出端口都对当前的仿真步有效，也即是说只有在一个模块的输出被更新之后，它才能作为其他模块的有效输入，去影响那些模块的行为。

(4) 更新模块当前主时间步的离散状态。在这个仿真阶段中，所有的模块都要进行每个时间步一次的活动——为当前时间的仿真循环更新离散状态。

(5) 积分。这个阶段只有模块具有连续状态或者非采样过零点时，才会存在。如果它是 S-函数，那么 Simulink 按最小时间步来调用 S-函数的输出和微分 S-函数方法。如果 S-函数具有非采样过零点（只有 C-MEX 函数才有可能存在这种情况），类似的，Simulink 按最小时间步来调用 S-函数中的输出和过零点部分，以便 Simulink 确定过零点的位置。

### 9.1.3 M 文件和 C MEX 文件 S-函数综述

在 M 文件 S-函数中，S-函数方法是用 M 文件子函数来实现的，而在 C MEX 文件 S-函数中，它们是用 C 函数来实现。M 文件 S-函数能使用的 S-函数方法都可以在 C MEX 文件 S-函数中找到对等的函数。但反过来不成立，Simulink 为 C MEXS-函数提供更多的方法。在 M 文件和 C MEX 文件 S-函数中都存在的方法，在两种文件里具有相同的名称。

对于 M 文件 S-函数，Simulink 通过传递一个 flag 参量给 S-函数，告诉 S-函数当前所处的仿真阶段，以便执行相应的子函数。于是，在写 M 文件 S-函数，用户只需用 MATLAB 语言为每个 flag 对应的 S-函数方法来编写代码即可。而在 C MEX 文件 S 函数中，Simulink 直接调用相应的 S-函数方法，而不是通过额外的参量（见后面的 C MEX 文件的编写）。表 9-1 列出了仿真阶段各自对应的 S-函数方法和 M 文件 S-函数中与它们相对应的 flag 值。

表 9-1 各个仿真阶段对应的方法

仿 真 阶 段	S-函数方法	flag (M 文件 S-函数)
初始化	mdlInitializeSizes	0
下一个采样点的计算（附加）	mdlGetTimeOfNextVarHit	4
输出值的计算	mdlOutputs	3
更新离散状态	mdlUpdate	2
微分的计算	mdlDerivatives	1
仿真任务的结束	mdlTerminate	9

C MEX S-函数里，S-函数方法的名称必须和表中的一样。但在 M 文件 S-函数中，S-函数方法（子函数）的名称不一定非得是表中的名称。读者可以用 switch 语句为每个 flag 值规定与它对应的 M 文件子函数的名称。为了便于用户编写 S-函数，Simulink 提供了一个名为 sfuntmpl.m 的模板文件，读者可以在 matlabroot /toolbox /Simulink /blocks 里找到它。使用模板文件写 M 文件函数，用户只需把自己的代码放到相应的 S-函数方法中去即可。而对于 C MEX S-函数，Simulink 同样提供了一个模板文件，它的名称是 sfuntmpl.c，所处的目录是 Simulink/src。

#### 9.1.4 S-函数概念

理解下面的这些关键概念，将有助于读者正确的建立 S-函数：

- (1) 直接馈入；
- (2) 宽度动态可变的输入信号；
- (3) 设置采样时间和偏移。

##### 1. 直接馈入

正如前面所讲过的，直接馈入指模块的输出或采样时间（变速率模型）直接由它的某个输入端口来控制。判断一个 S-函数是否具有直接馈入的标准有：

(1) 输出函数（mdlOutputs 或者 flag=3）是一个输入参数包含  $u$ （从 S-函数的角度）的函数，也就是计算系统输出的方程里包含变量  $u$  时，这个 S-函数就被认为是具有直接馈入。这一点和前面的论述是一致的。

(2) 如果该 S-函数是一个采样时间可变的 S-函数，并且下一个采样点的计算要求用到输入参数  $u$  时，也认为它是具有直接馈入的。

一个具有直接馈入的例子就是  $y=k \times u$  这个公式，其中  $k$  是增益， $u$  是输入信号， $y$  是输出。而不具有直接馈入的例子是下面公式表达的简单积分算法。

输出：  $y = x$

微分：  $x' = u$

其中， $x$  是状态值，而  $x'$  是状态对时间的微分， $y$  表示输出。

也许初学者，会问当传入的  $u$  是以前的输入值时，输出函数里包含  $u$  也是直接馈入吗？其实这种问题是不存在的，因为 Simulink 传给 S-函数的  $u$  都是当前时刻的输入值。如果 S-函数的某个子函数要用到以前的输入，那也只能用状态变量  $x$  来传递，当然，要先把输入存储到状态向量。因此，前面说的规则和模块的直接馈入的定义是不矛盾的。

##### 2. 宽度动态可变的输入信号

S-函数可以被写成支持任意输入宽度。这种情况下，输入信号的实际宽度可以在仿真开始阶段，通过 size 或者 length 等函数来求出输入向量的宽度。然后，就可以利用这个宽度来估计连续状态数目、离散状态数目和输出向量的宽度。

关于 S-函数的输入，还有一点需要读者注意。我们知道 Simulink 的内置模块的输入往往是分成多个端口，而每个端口又可以分别由若干个元素组成。这个特性的实现，在 M 文件 S-函数和 C MEX S-函数是有差别的。粗略地说，M 文件 S-函数对端口不加区分，它把所有端口合成一个输入向量；而 C MEX S-函数允许用户设置输入的端口数，然后再设置每个端口的元素个数。更详细的描述，读者可以在关于这两种 S-函数的编写具体介绍中找到。



### 3. 设置采样时间和偏移

M 文件和 C MEX S-函数都允许用户十分方便地设定 S-函数被调用的时间，也就是设置采样时刻。Simulink 为采样时间提供了下面的几种选择：

(1) 连续的采样时间——适用于具有连续状态和非采样过零点的 S-函数，这种 S-函数，输出按最小时间步改变。

(2) 连续的，但在最小仿真步具有固定值的采样时间——适用于需要在每一个主仿真时间步执行，但在最小仿真步内值不改变的 S-函数。

(3) 离散采样时间——如果 S-函数的行为的发生具有离散时间间隔的函数，用户可以定义一个采样时间来规定 Simulink 何时调用函数。而且用户还可以定义一个延迟时间 *offset* 来延迟采样点，注意 *offset* 的值不能超过采样周期，也就是相邻采样点的时间间隔。

一个采样点对应的的时间值由下面公式决定：

$$TimeHit = (n \times period) + offset$$

其中，*n* 是一个整数，它表示当前的仿真步，起始值总为 1。

如果用户定义了一个离散采样时间，那么 Simulink 就会在所定义在每个采样点调用 S-函数的 *mdlOutput* 和 *mdlUpdate* 方法。

(4) 可变采样时间——相邻采样点的时间间隔可变的离散采样时间。这种采样时间制式下，S-函数要在每一个仿真步的开始，计算下一个采样点的时刻。

(5) 继承的采样时间——有时候，一个 S-函数模块自身没有特定的采样时间，而它属于连续还是离散采样时间，完全取决于系统中的其他模块的采样时间。编写描述这种特性的模块的 S-函数时，可以把采样时间设定为继承。例如，*gain* 模块就是一个继承输入信号的采样时间的例子。一个模块可以从以下几种方式来继承采样时间：

- 继承驱动模块的采样时间；
- 继承目的模块的采样时间；
- 继承系统中的最快采样时间。

把采样时间设为继承的，要把 *sample time* 的值置为 -1。S-函数可以是单速率的也可以是多速率的。多速率的 S-函数具有多个采样时间。

采样时间按右边的格式成对说明：*[sample\_time, offset\_time]*。以下是几个有效采样时间。

*[CONTINUOUS\_SAMPLE\_TIME, 0.0]*

*[CONTINUOUS\_SAMPLE\_TIME, FIXED\_IN\_MINOR\_STEP\_OFFSET]*

*[discrete\_sample\_time\_period, offset]*

*[VARIABLE\_SAMPLE\_TIME, 0.0]*

其中，斜体书写的变量表示用户在设定时要用具体实数去替代它们，而字母大写的字符串是几个 Simulink 定义过的常量，它们的取值分别为：

*CONTINUOUS\_SAMPLE\_TIME* = 0.0

*FIXED\_IN\_MINOR\_STEP\_OFFSET* = 1.0

*VARIABLE\_SAMPLE\_TIME* = -2.0

同样，用户可以设置采样时间从驱动模块继承，这时采样时间只能有一个采样时间对。

[INHERITED\_SAMPLE\_TIME, 0.0]

[INHERITED\_SAMPLE\_TIME, FIXED\_IN\_MINOR\_STEP\_OFFSET]

其中，

INHERITED\_SAMPLE\_TIME = -1.0

下面的几个准则可以帮助用户说明采样时间。

(1) 在最小积分步值会变化的连续 S-函数，采样时间应该设置为[ CONTINUOUS\_SAMPLE\_TIME, 0.0 ]。

(2) 连续但在最小积分步值不变化的 S-函数，应该具有[CONTINUOUS\_SAMPLE\_TIME, FIXED\_IN\_MINOR\_STEP\_OFFSET]的采样时间。

(3) 以一个固定速率变化的离散 S-函数应该具有离散时间对，[discrete\_sample\_time\_period, offset]。这里

$$\text{discrete\_sample\_period} > 0.0$$

以及

$$0.0 < \text{offset} < \text{discrete\_sample\_period}$$

(4) 以变速率执行的离散 S-函数，应该具有可变步长的离散采样时间：[ VARIABLE\_SAMPLE\_TIME, 0.0]。

如果用户的 S-函数具有继承的采样时间，存在两种情况：

- 值随输入信号变化而变化，甚至在最小的积分步也是如此的 S-函数，应该设置采样时间为[INHERITED\_SAMPLE\_TIME, 0.0]。

- 值随输入信号变化而变化，但在最小的积分步不变化的 S-函数，应该设置采样时间为[INHERITED\_SAMPLE\_TIME, FIXED\_IN\_MINOR\_STEP\_OFFSET]。

## 9.2 建立 M 文件 S-函数

定义 S-函数的 M 文件要提供模型有关的信息，这些信息是仿真过程中为 Simulink 所必须的。仿真进行时，Simulink、ODE 解法器和 M 文件交互来完成特定的任务。这些任务包括定义初始条件和模块特性、计算微分、离散状态和输出。

### 9.2.1 如何使用模板

建立 M 文件 S-函数的推荐方法是使用 Simulink 提供的模板 M 文件——sfuntmpl.m，它的位置在 MATLAB 根目录下的 toolbox/ Simulink/ blocks 目录。读者可以用 MATLAB Edit 浏览这个模板文件。它里面的代码如下所示：

```
function [sys,x0,str,ts] = sfuntmpl (t, x, u, flag)
```

```
switch flag,
    case 0,
        [sys,x0,str,ts]=mdlInitializeSizes;
    case 1,
        sys=mdlDerivatives (t,x,u) ;
    case 2,
        sys=mdlUpdate (t,x,u) ;
    case 3,
        sys=mdlOutputs (t,x,u) ;
    case 4,
        sys=mdlGetTimeOfNextVarHit (t,x,u) ;
    case 9,
        sys=mdlTerminate (t,x,u) ;
    otherwise
        error (['Unhandled flag = ',num2str (flag) ]) ;
end
% end sfuntmpl
```

```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 0;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed
sys = simsizes ( sizes ) ;
x0 = [];
str = [];
ts = [0 0];
% end mdlInitializeSizes
```

```
% mdlDerivatives
function sys=mdlDerivatives (t,x,u)
sys = [];
% end mdlDerivatives
```

```
function sys=mdlUpdate (t,x,u)
sys = [];
```

```

% end mdlUpdate

function sys=mdlOutputs (t,x,u)

sys = [];

% end mdlOutputs


function sys=mdlGetTimeOfNextVarHit (t,x,u)

sampleTime = 1;    % Example, set the next hit to be one second later.

sys = t + sampleTime;

% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate (t,x,u)

sys = [];

% end mdlTerminate

```

为了方便叙述，摘取时把主要的注释文字去掉了。下面就来简略地讲解它的使用。

模板文件里 S-函数的结构十分简单，它只为不同的 flag 的值指定要相应调用的 M 文件子函数，例如当 flag 值为 3 时，即模块处于计算输出这个仿真阶段时，相应调用的子函数为 `sys= mdlOutputs (t, x, u)`；模板文件使用 `switch` 语句来完成这种指定，当然这种结构并不唯一，读者完全可以使用 `if` 语句来完成同样的功能。而且在实际运用时，可以根据实际需要来去掉某些值，因为并不是每个模块都要经过所有的仿真阶段。

读者要明白一点，模板文件只是 Simulink 为方便用户而提供的一种参考格式，并不是编写 S-函数的语法要求。例如读者完全可以把子函数取成别的名字，或者直接把代码写在主函数里。但使用模板的一个好处，就是比较方便，而且条理清晰。

使用模板编写 S-函数，用户只需把 S-函数名换成期望的函数名称，如果需要额外的输入参量，还需在输入参数列表的后面增加这些参数，因为前面的 4 个参数是 Simulink 调用 S-函数时自动传入的。而对于输出参数，最好不要做任何修改。主函数基本上就可以不用读者再进行别的修改了。接下去的工作就是根据所编 S-函数要完成的任务，用相应的代码去替代模板里各个子函数的代码，因为模板里最初的代码实际上什么也没有做。

下面就来介绍完成各个仿真阶段的子函数的编写方法。

首先，来解决一个容易让 S-函数新手困惑的问题。无论是在哪个仿真阶段，相应的 S-函数方法的返回变量都是 `sys`，就像模板 M 文件显示的一样，计算得到的输出、更新好的离散状态都是由 `sys` 变量返回。要弄清楚这个问题，还是要回到 Simulink 如何调用 S-函数上来。前面讲过，Simulink 在每个仿真步的仿真循环中的每个仿真阶段都会对 S-函数进行调用。在调用时，Simulink 不但根据所处的仿真阶段为 flag 传入不同的值，而且还会为 `sys` 这个返回变量指定不同的角色，也就是说尽管是相同的 `sys` 变量，但在不同的仿真阶段其意义却不相同，这种变化由 Simulink 自动完成。

表 9-2 列出了 M 文件 S-函数可用的 S-函数方法。

表 9-2 M 文件 S-函数方法

S-函数方法	说 明
mdlInitializesizes	定义 S-函数模块的基本特性，包括采样时间、连续或者离散状态的初始条件和 sizes 数组
mdlDerivatives	计算连续状态变量的微分
mdlUpdate	更新离散状态、采样时间和主时间步的要求
mdlOutputs	计算 S-函数的输出
mdlGetTimeOfNextVarHit	计算下一个采样点的绝对时间，这个方法仅仅是在用户在 mdlInitializeSizes 说明了一个可变的离散采样时间
mdlTerminate	实现仿真任务必须的结束

概括地说来，建立 S-函数可以被当成两个分离的任务：

- （1）初始化模块特性包括输入输出信号的宽度，离散连续状态的初始条件和采样时间。
- （2）将算法放到合适的 S-函数方法中去。

9.2.2 定义 S-函数的初始信息

为了让 Simulink 识别出一个 M 文件 S-函数，用户必须在 S-函数里提供有关 S-函数的说明信息，包括采样时间、连续或者离散状态个数等初始条件。这一部分主要是在 mdlInitializesizes 方法里完成，请读者对照模板文件，来学习它的代码编写。在所有的 M 文件 S-函数方法编写中，这个方法的语法约束应该是最多的。

读者可以看到在模板文件的 mdlInitializesizes 方法中，首先出现的语句是

```
sizes = simsizes;
```

这个语句返回一个没有经过初始化的 sizes 结构，sizes 结构是 S-函数信息的载体，它内部的字段，可以在 MATLAB 命令窗口查询：

```
>>sizes = simsizes;
>>sizes
sizes =
    NumContStates:    0
    NumDiscStates:    0
    NumOutputs:       0
    NumInputs:        0
    DirFeedthrough:   0
    NumSampleTimes:   0
```

上面的结果反映了 sizes 是一个具有 6 个字段的结构，它里面字段的值在开始时都被置为零。往 sizes 结构载入 S-函数的说明信息，就是重置这些字段的值。表 9-3 列出了这些字段的意义。

表 9-3 sizes 各字段的意义

字 段 名	说 明
sizes.NumContStates	连续状态的个数（状态向量连续部分的宽度）
sizes.NumDiscStates	离散状态的个数（状态向量离散部分的宽度）
sizes.NumOutputs	输出的数目个数（输出向量的宽度）
sizes.NumInputs	输入的数目个数（输入向量的宽度）
sizes.DirFeedthrough	有无直接馈入
sizes.NumSampleTimes	采样时间的个数

表中的这些字段的意义非常明确，在建立 S-函数时，我们可以很方便的根据 S-函数的情况来对这些字段赋值。这里还是以前面曾提过的 S-函数 `timestwo` 来举例。它的 `mdlInitializesizes` 的代码如下：

```
function [sys, x0, str, ts] = mdlInitializeSizes ( )

sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 0;
sizes.NumOutputs = -1; % 动态宽度的输出
sizes.NumInputs = -1; % 动态宽度的输入
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes (sizes) ;
str = [];
x0 = [];
ts = [-1 0]; % inherited sample time
% end mdlInitializeSizes
```

这个 S-函数的作用就是把输入信号乘以 2 倍，只需直接对输入信号操作即可，而不需要任何的状态变量来保存系统的一些内部状况（关于状态的理解，本章后面会用一个例子来说明）。因此，在说明 S-函数信息时，连续状态数和离散状态数都被赋值为 0。而对于函数的输入向量和输出向量的宽度，由于是受驱动模块控制的，所以属于动态可变，为此，要把 `NumOutputs` 和 `NumInputs` 赋为 -1 来说明它们是动态可变的。如果输入输出的宽度是固定的，那么就要给这两个字段赋上实际的宽度值。`sizes` 结构的前面 4 个字段其实都是可以赋值为 -1，来表示它们的值是动态可变的。

要注意 `DirFeedthrough` 是一个布尔变量，它的取值只有 0 和 1 两种，0 表示没有直接馈入，1 表示有直接馈入。如果 `DirFeedthrough` 的值被赋为 0，那么读者在编写 `mdlOutputs` 时

就要确保子函数的代码里不出现输入变量  $u$ ，否则会出现无法预计的结果。

`NumSampleTimes` 表示采样时间的个数，也就是 `ts` 变量的行数，与用户对 `ts` 的定义有关。

对 `sizes` 赋值完毕后，再调用 `simsizes`，这时要传入定义好的 `sizes` 结构作为参数，并把结果返回给 `sys` 变量，相应的语句是：

```
sys = simsizes (sizes);
```

其实，`simsizes (sizes)` 只是把 `sizes` 结构组合成一个长度为 6 的向量。下面的命令证明了这个事实。

```
>> sizes.NumContStates=1;
```

```
>> sys = simsizes (sizes);
```

```
>> sys
```

```
sys =
```

```
1    0    0    0    0    0    0
```

所以，完全可以不需要前面那么一大段的步骤，而直接对 `sys` 进行赋值，例如 `timestwo` 的 `sys` 变量就直接可以写为：

```
>> sys=[0, 0, -1, -1, 1, 1];
```

但提醒读者，赋值时一定要小心，避免出现输入错误。

在这个小节的最后，我们再来强调一下如何确定 S-函数的输入、输出个数。例如，要编写一个描述将两个输入信号相加的 S-函数。封装好的 S-函数模块在外形上就应该有两个输入端口，而每个端口又有多个元素。前面曾讲过，S-函数会忽略端口。事实上，S-函数的输入个数被设为所有端口的元素个数的总和，而不是端口的个数，也就是说所有的输入端口都合并到一个输入向量中。但是模块是通过两个端口来输入，于是在使用 S-函数时，要用一个 `Mux` 模块将两个输入信号合并成一个信号再连到 S-function 模块。

### 9.2.3 输入和输出参量说明

S-函数前面的四个输入参量，必须是 `t`、`x`、`u` 和 `flag`，并且次序不能变动。它们的意义分别是：

(1) `t` 代表当前的仿真时间，这个输入参数通常用于决定下一个采样时刻，或者在多采样速率系统中，来区分不同的采样时刻点，并据此进行不同的处理。

(2) `x` 表示状态向量，这个参数是必需的，甚至在系统中不存在状态时也是如此。状态有很灵活的运用，本章后面会用一个例子来说明。

(3) `u` 表示输入向量。

(4) `flag` 是一个控制在每一个仿真阶段调用哪种 S-函数方法的参数，由 `Simulink` 在调用时自动取值。

`Simulink` 也需要四个返回参数——`sys`、`x0`、`str` 和 `ts`，它们的排列顺序也必须是模板所指定的顺序。这些参数的意义为：

(1) `sys` 是一个通用的返回参数，它所返回值的意义取决于 `flag` 的值，例如，当 `flag` 的值是 3 时，`sys` 包含的是 S-函数的输出。

(2) `x0` 是初始的状态值（没有状态时，就是一个空矩阵[]），这个返回参数只在 `flag` 值为 0 时才有效，其他时候都会被忽略。

(3) `str` 这个参数没什么意义，是 MathWorks 公司为将来的应用保留的，M 文件 S-函数必须把它设为空矩阵[]。

(4) `ts` 是一个  $m \times 2$  的矩阵，它的两列分别表示采样时间间隔和偏移。关于不同的采样时间，如何进行设置，在前面已经提到过，这里就不再赘述。`ts` 可以被赋为一个多行的矩阵，这时矩阵所有行的采样时间必须按单调递增的顺序排列。例如，读者想让 S-函数在 `[0, 0.1, 0.25, 0.75, 1.0, 1.1, 1.25, ...]` 执行，可以为 `ts` 设置  $2 \times 2$  的矩阵，即

```
ts = [.25, 0; 1.0, .1]; % 多采样速率的设置
```

### 9.2.4 M 文件 S-函数的几个示例

下面分别就连续状态系统、离散状态系统、混合系统和变步长系统各自举一个 S 函数例子，来加深对用 M 文件编程的理解。最后，还举了一个移位寄存器序列的例子来加深对状态向量的理解。

#### 1. 连续状态 S-函数示例

这个例程定义了一个连续状态空间系统，它的状态方程为

$$\begin{aligned} \dot{x}' &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

从这个例子，读者应该强化的几个概念是：如何在初始化方法里，定义 S-函数的有关信息，如何根据所描述的系统来确定连续和离散状态数，输入和输出信号数。在这个例子中不存在离散状态，所以离散状态数就被设置为 0，而连续状态向量（微分方程所对应的）的宽度为 2，所以连续状态数就是 2。而对于输入和输出，它们的信号宽度都是 2，所以相应的描述参数也被设为 2。这里要记住一点，输入信号数 `sizes.NumOutputs` 和输出信号数 `sizes.Numinputs`，指的是信号的宽度，而不是模块里的端口个数，也就是说，它们是模块里所有输入端口（输出端口）的信号宽度之和。

还应该记住的一点是，S-函数方法 `mdlDerivatives` 返回的是连续状态的微分值，Simulink 会把它作为微分值来处理，从而计算连续状态的积分。

在例子中，状态空间方程中的参数 `A`、`B`、`C`、`D` 都是作为固定值在程序中指定的，其实，完全可以把它们作为参数由使用者在外面设定，就像 Simulink 模块 `State Space` 模块所表现的那样。这时，就要根据这些参数在 `mdlInitializeSizes` 方法动态的设定各个函数的描述信息值。实现的方法很容易想出，不清楚的读者可以参考本节的最后一个例程。

下面是这个例程的代码（例 1 至例 4 的 M 文件 S-函数代码可以在 `simulink/blocks` 目录里找到，在这个目录里，还提供了其他的一些示例）：

```
function [sys,x0,str,ts] = csfunc (t,x,u,flag)
%CSFUNC An example M-file S-function for defining a continuous system.
```



```
% Example M-file S-function implementing continuous equations:
%      x' = Ax + Bu
%      y  = Cx + Du
% See sfuntmpl.m for a general S-function template.
% See also SFUNTMPL.

% Copyright (c) 1990-1998 by The MathWorks, Inc. All Rights Reserved.
% $Revision: 1.5 $
```

```
                % 定义四个参数%
A=[-0.09   -0.01
    1       0];

B=[ 1   -7
    0   -2];

C=[ 0    2
    1   -5];

D=[-3    0
    1    0];

switch flag,

case 0,
    [sys,x0,str,ts]=mdlInitializeSizes (A,B,C,D) ;
                %初始化 %

case 1,
    sys=mdlDerivatives (t,x,u,A,B,C,D) ;
                % 计算微分 %

case 3,
    sys=mdlOutputs (t,x,u,A,B,C,D) ;
                % 计算输出 %

case { 2, 4, 9 },
    sys = [];
                % 不需要的 flag 值%

otherwise
    error (['Unhandled flag = ',num2str (flag) ] ) ;

end
```

```

% end csfunc
% mdlInitializeSizes

function [sys,x0,str,ts]=mdlInitializeSizes (A,B,C,D)

sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
    % 当矩阵 D 不是零矩阵时，函数具有直接馈入
sizes.NumSampleTimes = 1;

sys = simsizes (sizes) ;
x0 = zeros (2,1) ;
str = [];
ts = [0 0];
    % 设置连续的采样时间
% end mdlInitializeSizes
% mdlDerivatives

function sys=mdlDerivatives (t,x,u,A,B,C,D)

sys = A*x + B*u;

% end mdlDerivatives

% mdlOutputs

function sys=mdlOutputs (t,x,u,A,B,C,D)

sys = C*x + D*u;

% end mdlOutputs

```

## 2. 离散状态 S-函数示例

这个例程则定义了一个离散线性系统，它实现的离散方程是：

$$x(n+1) = Ax(n) + Cu(n)$$

$$y(n) = Cx(n) + Du(n)$$

编写离散状态 S-函数，需要调用对状态进行更新的 S-函数方法 mdlUpdates。其余的地方和连续状态 S-函数的编写区别不大。整个文件的代码如下：

```
function [sys,x0,str,ts] = dsfunc (t,x,u,flag)
% An example M-file S-function for defining a discrete system.
% This S-function implements discrete equations in this form:
% x(n+1)= Ax(n) + Bu(n)
% y(n)= Cx(n) + Du(n)
% Generate a discrete linear system:
% 定义四个参数%
A=[-1.3839 -0.5097
1.0000 0];
B=[-2.5559 0
0 4.2382];
C=[ 0 2.0761
0 7.7891];
D=[ -0.8141 -2.9334
1.2426 0];
switch flag,
case 0
    sys = mdlInitializeSizes (A,B,C,D) ;
case 2
    sys = mdlUpdate (t,x,u,A,B,C,D) ;
case 3
    sys = mdlOutputs (t,x,u,A,B,C,D) ;
case {1, 4, 9} % Unused flags
    sys = [];
otherwise
    error ( ['unhandled flag = ',num2str (flag) ] ) ; % Error handling
end
% End of dsfunc.

% Initialization

function [sys,x0,str,ts] = mdlInitializeSizes (A,B,C,D)
sizes = simsizes;
```

```

sizes.NumContStates = 0;
sizes.NumDiscStates = 2;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1; %当矩阵 D 不为空
sizes.NumSampleTimes = 1;
sys = simsizes ( sizes );
x0 = ones ( 2,1 ); % 初始化离散状态
str = [];
ts = [1 0]; % 采样时间: [period, offset]
% End of mdlInitializeSizes.

```

```

% mdlUpdates

```

```

function sys = mdlUpdates (t,x,u,A,B,C,D)
sys = A*x + B*u;
% End of mdlUpdate.

```

```

% Calculate outputs

```

```

function sys = mdlOutputs (t,x,u,A,B,C,D)
sys = C*x + D*u;
% End of mdlOutputs.

```

### 3. 混合系统 S-函数

这个 M 文件定义了一个描述混合系统（连续状态和离散状态的结合）的 S-函数。这个 S-函数所描述的系统等价于一个积分器后接一个离散单位时间延迟的混合系统（图 9-5）。

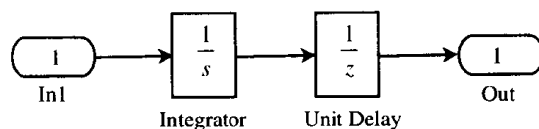


图 9-5 混合系统示例

在混合系统中，与离散状态和连续状态有关的 S-函数方法，都有可能执行，但显然各自的执行时间是不一样的。如积分器在每一个微时间步都会执行，而单位时间延迟只在离散采样时刻到达时执行。

下面就简要的描述一下这个例程的编写要点。首先，例程在 `mdlInitializeSizes` 方法里定义了两个采样时间：连续采样时间和由离散采样周期决定的离散采样周期。于是在这两个采样时间决定的采样点，`Simulink` 都会调用 S-函数文件，并依次用 `flag` 来指定要调用仿真循环

中的哪个方法。

我们知道，尽管连续状态的微分计算是在每一个微时间步都要进行的，但离散状态的更新以及系统的输出的产生，只是在离散采样点到达时才完成。但这一点 Simulink 是无法区分什么时候该更新什么时候不该更新。因为 Simulink 从初始化获得的信息只是，该模型既有离散状态又有连续状态，据此它会决定该函数的仿真循环应该包含连续状态的微分计算和离散状态更新这两个方法，而没有足够的信息来判断当前时刻是不是离散采样点。

这时常用的一个技巧是，在 mdlUpdate 和 mdlOutputs 中由程序显示判断当前时刻是不是离散采样点。完成它的语句是：

```
if abs (round ( (t-doffset) /dperiod) - (t-doffset) /dperiod) < 1e-8
    sys = x (1);    % 表示是离散采样点，就把
                    % 离散状态更新为当前的连续状态
else
    sys = [];       % 否则离散状态就不改变。
end
```

mdlOutputs 方法中的判断语句也是极为类似的。整个文件的代码如下：

```
function [sys,x0,str,ts] = mixedm (t,x,u,flag)
% A hybrid system example that implements a hybrid system
% consisting of a continuous integrator (1/s) in series with a
% unit delay (1/z) .
%
% Set the sampling period and offset for unit delay.
dperiod = 1;
doffset = 0;
switch flag,
    case 0
        [sys,x0,str,ts] = mdlInitializeSizes (dperiod,doffset);
    case 1
        sys = mdlDerivatives (t,x,u);
    case 2
        sys = mdlUpdate (t,x,u,dperiod,doffset);
    case 3
        sys = mdlOutputs (t,x,u,doffset,dperiod);
    case {4,9}
        sys = []; % Unused flags
    otherwise
        error (['unhandled flag = ',num2str (flag)]); % Error handling
end
% End of mixedm.
```

```

%
% mdlInitializeSizes

function [sys,x0,str,ts] = mdlInitializeSizes ( dperiod,doffset)
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 1;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 2;
sys = simsizes (sizes) ;
x0 = ones (2,1) ;
str = [];
ts = [0, 0;  dperiod, doffset];
% End of mdlInitializeSizes.

% mdlDerivatives

function sys = mdlDerivatives (t,x,u)
sys = u;
% end of mdlDerivatives.

% mdlUpdate

function sys = mdlUpdate (t,x,u,dperiod,doffset)
if abs (round ( (t-doffset) /dperiod) - (t-doffset) /dperiod) < 1e-8
    sys = x (1) ;
    % 是一个离散采样点，更新
    % 离散状态值为当前连续状态值
else
    sys = [];
    % 表示不是一个离散采样点，
    % 返回一个空矩阵，表示没有变化
end
% End of mdlUpdate.

% mdlOutputs

```

```

function sys = mdlOutputs (t,x,u,doffset,dperiod)
if abs (round ( (t-doffset) /dperiod) - (t-doffset) /dperiod) < 1e-8
    sys = x (2);
else
    sys = [];
    % 表示不是一个离散采样点,
    % 返回一个空矩阵, 表示没有变化
end

```

#### 4. 可变步长仿真系统示例

这是一个可变仿真步长系统的 S-函数示例, 也就是说仿真的步长在运行过程中可以动态变化。就像本例中, S-函数实现对输入的第一个元素的延迟输出, 而延迟的时间由输入的第 2 个元素决定。在 S-函数中, 实现这一点主要使用 `mdlGetTimeOfNextVarHit` 方法, 它的作用是决定下一个采样点的时间, 对应的 `flag` 参数是 4。要使 Simulink 在仿真循环中调用这个方法, 需先在 `mdlInitializeSizes` 方法里说明采样时间是变采样时间, 设置方法如下

```
ts = [-2, 0];
```

在初始化模块信息时, 还要注意直接馈入的设置。因为在 S-函数中采样时间的计算和输入直接有关, 所以这个 S-函数是具有直接馈入的。

初始化信息之后, 就可以在 `mdlGetTimeOfNextVarHit` 方法里计算出下一个采样时间。其代码是

```
sys = t + u (2);
```

注意, 返回值同样是传给 `sys` 变量, 而不要犯  $t = t + u(2)$  的错误。

下面是这个例子的源代码:

```

function [sys,x0,str,ts] = vsfunc (t,x,u,flag)
% This example S-function illustrates how to create a variable
% step block in Simulink. This block implements a variable step
% delay in which the first input is delayed by an amount of time
% determined by the second input:
% dt = u (2)
% y (t+dt) = u (t)
%
switch flag,
case 0
    [sys,x0,str,ts] = mdlInitializeSizes; % 初始化
case 2
    sys = mdlUpdate (t,x,u); % 更新离散状态
case 3

```

```

    sys = mdlOutputs (t,x,u) ; % 计算输出
case 4
    sys = mdlGetTimeOfNextVarHit (t,x,u) ; % 获得下一个采样时间点
case { 1, 9 }
    sys = []; % 无用的 flag
otherwise
    error (['Unhandled flag = ',num2str (flag) ]) ; % Error handling
end
% End of vsfunc.

```

```

% mdlInitializeSizes

```

```

function [sys,x0,str,ts] = mdlInitializeSizes

```

```

    sizes = simsizes;
    sizes.NumContStates = 0;
    sizes.NumDiscStates = 1;
    sizes.NumOutputs = 1;
    sizes.NumInputs = 2;
    sizes.DirFeedthrough = 1; % 因为下一个采样时间的确定和
                                % 输入有关，所以需要直接馈入
    sizes.NumSampleTimes = 1;
    sys = simsizes (sizes) ;

```

```

% 初始化初始条件

```

```

    x0 = [0];
    str = [];
    ts = [-2 0]; % 定义采样时间为可变的
% End of mdlInitializeSizes.

```

```

% mdlUpdate

```

```

function sys = mdlUpdate (t,x,u)
    sys = u (1) ;
% End of mdlUpdate.

```

```

% mdlOutputs

```



```
function sys = mdlOutputs (t,x,u)
```

```
sys = x (1);
```

```
% end mdlOutputs
```

```
% mdlGetTimeOfNextVarHit
```

```
function sys = mdlGetTimeOfNextVarHit (t,x,u)
```

```
sys = t + u (2);
```

```
% End of mdlGetTimeOfNextVarHit.
```

### 5. 最大长度线性移位寄存器序列的生成

这个例子主要是为了加深对状态的理解。前面的那些例子中的状态概念有浓厚的信号与系统理论背景。但从编程的角度讲，状态也可以理解为一个全局变量，函数编写者可以用状态（离散状态）来保留一些全局信息。

本例描述的一个线性移位寄存器序列发生器系统，图 9-6 显示的一个。这里不讨论这个系统的理论背景，感兴趣的读者请参阅有关的书籍。

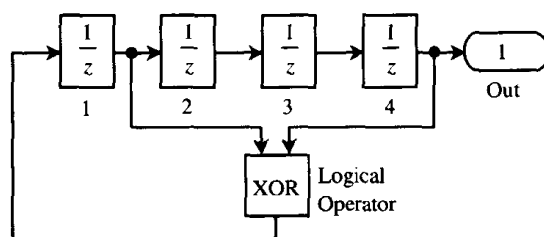


图 9-6 一个 4 级移位寄存器序列

从图中，读者可以看出，系统由一系列单位时延模块串接而成，模块的个数也称为级数。而这些模块中某些输出被连到一个异或逻辑运算模块进行异或（模 2 加），得出的结果再输出到第一个时延的输入，最后一个时延模块的输出作为系统的输出。在每个时延模块的初始状态设为 0 或 1 情况下，执行模型，就可以得到一个 0、1 二进制序列。

至于每个模块是否连接到异或模块进行运算的状态，可以用一个向量 **poly** 来表示。对于  $m$  级系统，向量 **poly**=[c1; c2; c3; c4; ...; cm]，它的元素要么为 0，要么为 1。如果第  $i$  个元素为 0 表示第  $i$  个单元不要连接到异或模块，为 1 则表示要连接。例如图 9-6 演示的系统对应的向量为[1; 0; 0; 1]；

下面的 S-函数将 **poly**、单位延迟时间（**period**）和初始状态向量（**ini\_st**）作为额外参数，这样就可以让使用者在外面来设置这些参数，生成不同级数，不同连接情况，不同周期和不同初始状态的移位寄存器序列。

该函数，读者要注意如何在初始化时，根据传入的参数来动态地变化 S-函数的有关信息，这里是离散状态数。

另外一点，本例是通过用一个长度和级数相同的离散状态向量来表示串接的单位延迟模块。在每个采样时刻，通过将状态向量左移位就可以表示单位延迟的串接。S-函数的代码如下所示。

```

function [sys,x0,str,ts] = pnsequence (t,x,u,flag,poly,period,ini_st)
switch flag
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes (poly,period,ini_st) ;
case 2,
    sys=mdlUpdate (t,x,u,poly) ;
case 3,
    sys=mdlOutputs (t,x,u) ;
case {1,4,9},
    sys=[];
otherwise
    error (['Unhandled flag = ',num2str (flag) ]) ;

end

% end sfuntmpl

% mdlInitializeSizes

function [sys,x0,str,ts]=mdlInitializeSizes (poly,period,ini_st)
n_dis=length (poly) ;
sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = n_dis;
sizes.NumOutputs = 1;
sizes.NumInputs = 0;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % at least one sample time is needed
sys = simsizes (sizes) ;
x0=[ini_st,0];
str = [];
ts = [period 0];
% end mdlInitializeSizes

%mdlUpdate
function sys=mdlUpdate (t,x,u,poly)
reg=x;
n_x=length (x) ;

```

```

temp=mod (poly*reg (1:n_x-1),2); % 完成异或
reg=[reg (2:n_x) ;temp]; % 进行左移位
sys =reg;

% end mdlUpdate

%mdlOutputs
function sys=mdlOutputs (t,x,u)
sys = x (1);

% end mdlOutputs

```

## 9.3 C MEX S-函数简介

M 文件 S-函数的优点是编写简单，但它会影响仿真的运行速度，而且包含 M 文件 S 函数的模型无法生成实时代码，无法利用 RTW 提供的许多强大功能（见第十章）。所以，如果要想利用 S-函数高效地对 Simulink 进行扩展，那么就必须掌握 C MEX S-函数的编写方法。这里只对其作一个简单的介绍，更详细的信息，请读者参阅 Simulink 的帮助文档。

### 9.3.1 介绍

同 M 文件 S-函数一样，在仿真时，一个 S-函数模块必须提供给 Simulink 有关的模型信息。当仿真进行时，Simulink、ODE 解法器和 MEX 文件交互地实现指定的任务。这些任务包括定义初始条件和模块特性、计算微分、离散状态和输出，它们的定义和 M 文件 S-函数的一样。

确切地说，C MEX S-函数有着和 M 文件 S-函数相同的结构，它能够实现 M 文件 S-函数能实现的功能。而且，C MEX S-函数比 M 文件 S-函数为用户提供更多的功能。与 M 文件 S-函数类似，Simulink 同样提供了模板文件 sfuntmpl.c 以及它的复杂版本 sfuntmpl.doc（都在 Simulink /src 目录下），来方便读者编写 C MEX S-函数。

C MEX S-函数源文件的通用格式如下所示：

```

#define S_FUNCTION_NAME your_sfunction_name_here %在这里定义 S-函数名
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
static void mdlInitializeSizes (SimStruct *S)
{
}
<additional S-function routines/code>
static void mdlTerminate (SimStruct *S)

```

```

{
}

#ifdef MATLAB_MEX_FILE /* 这个文件是编译为一个 MEX 文件吗? */
#include "Simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* 代码生成注册函数*/
#endif

```

`mdlInitializeSizes` 是 Simulink 在和 S-函数交互时首先调用的方法。在一个 S-函数源文件内，还有其他的几个 `mdl*` 方法。所有的 S-函数源文件必须和这个格式一致，也就是说不同的 S-函数文件内的方法可以使用相同的名称（如 `mdlInitializeSizes`），因为不同的 S-函数文件是通过 S-函数名称来区别的。在 Simulink 调用 `mdlInitializeSizes` 后，通过其他的几种方法（都以 `mdl` 开头）来和 S-函数实现交互，在仿真结束时，`mdlTerminate` 就被调用。

和 M 文件 S-函数不同，Simulink 不是通过显式的 `flag` 参数来指定调用何种 C MEX S-函数方法。这是因为 Simulink 在交互作用时，会自动地在合适的时候调用每个 S-函数方法。同样，C MEX S-函数中的某些方法在 M 文件 S-函数中，可以找到对应项。

Simulink 保存 S-函数的有关信息在一个名为 `SimStruct` 的数据结构中。宏语句 `#include "simstruc.h"` 定义了 `SimStruct`，它使用户的 MEX-文件能在 `SimStruct` 中赋值和从中取值。

### 9.3.2 编写基本的 C MEX S-函数

这一节讨论如何建立基本的 C MEX S-函数，这里的基本是指仅仅包含具有规定的 S-函数方法的 S-函数。然而，这些方法的内容可以任意的复杂。用户可以随意的把任何逻辑代码放入 S-函数方法，只要它符合规定的格式。这里，就以前面一节提过的 `timestwo` 的 C MEX S-函数版本 `doubleinput` 来作为示例。

学习过 M 文件 S-函数编写的读者，知道 S-函数通过 Simulink 提供的 S-Function 模块和 Simulink 模型合并起来使用。那么 C MEX S-函数呢？它同样也是通过这个模块。请读者看图 9-7 所示的模型。模型中，`doubleinput` S-函数的作用是将正弦波的幅度加倍，并且显示在 `scope` 上。

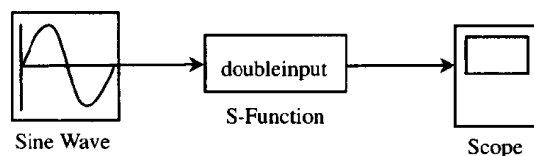


图 9-7 将 S-函数合并到 Simulink 模型中

S-Function 模块对话框的设置和 M 文件 S-函数一样，`name` 参数的值为 `doubleinput`，`parameters` 参数编辑框则空着不用填写。但 C MEX S-函数和 M 文件 S-函数不同的一点是，M 文件 S-函数编好之后直接就可以使用；但是 C MEX S-函数则不然，在 S-函数的源文件写好之后，还必须使用 `mex` 命令将其编译为 MEX 可执行文件。例如，在命令行输入

```
>> mex doubleinput.c
```

这个命令将使 MATLAB 编译和链接 doubleinput.c 文件，这将生成一个供 Simulink 使用的动态装载的可执行文件。这个可执行文件就被称为 MEX S-函数，其中 MEX 代表“MATLAB Executable”。MEX 文件的扩展名因平台而异，在微软视窗环境下，其扩展名就是.dll。

✎ mex 是 MATLAB 的编译命令，它通常把编译好的 MEX 文件保存在当前工作目录中，所以读者在使用该命令之前，最好是在路径浏览器里把当前目录设置为读者自己的工作目录。至于 mex 命令的详细说明，请读者自行查阅帮助。

图 9-8 显示了 doubleinput.c 所包含的 S-函数方法。

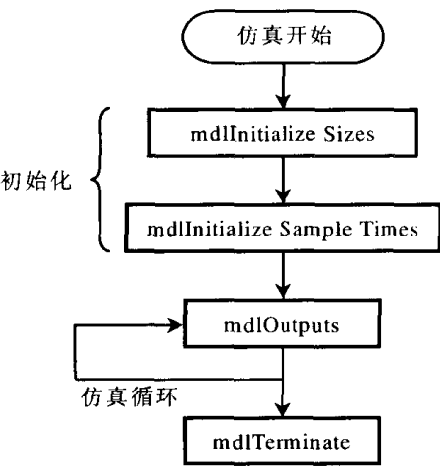


图 9-8 doubleinput.c 中的 S-函数方法

表 9-4 则列出了这些 S-函数方法的描述。它们的功能和 M 文件 S-函数中的对应方法大致相同。

表 9-4 S-函数方法描述

S-函数方法	说 明
mdlInitializeSizes	当 Simulink 开始处理模型并决定输入和输出端口的数目时，就调用这个方法。在仿真开始时 Simulink 同样会调用这个方法，以获得函数的信息，如端口的尺寸以及状态数（和 M 文件 S 函数类似）
mdlInitializeSampleTimes	Simulink 调用这个方法来说设置 S-函数的采样时间。doubleinput 有一个单一的继承采样时间，SAMPLE_TIME_INHERITED
mdlOutputs	计算输出。在本例中，将输入信号乘上 2，再放入输出信号中。这个方法在仿真循环中输出需要更新的每个时间步被调用，这里就是每个仿真时间步
mdlTerminate	实现在仿真结束时的操作。doubleinput 因为没有任何操作要进行，所以为空

doubleinput.c 的代码如下：

```
#define S_FUNCTION_NAME doubleinput
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

/* Function: mdlInitializeSizes
```

```

* Abstract:
*   Setup sizes of the various vectors.
*/
static void mdlInitializeSizes (SimStruct *S)
{
    ssSetNumSFcnParams (S, 0) ;
    if (ssGetNumSFcnParams (S) != ssGetSFcnParamsCount (S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    if (!ssSetNumInputPorts (S, 1)) return;
    ssSetInputPortWidth (S, 0, DYNAMICALLY_SIZED) ;
    ssSetInputPortDirectFeedThrough (S, 0, 1) ;
    if (!ssSetNumOutputPorts (S, 1)) return;
    ssSetOutputPortWidth (S, 0, DYNAMICALLY_SIZED) ;
    ssSetNumSampleTimes (S, 1) ;
    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions (S, SS_OPTION_EXCEPTION_FREE_CODE) ;
}

/* Function: mdlInitializeSampleTimes
* Abstract:
*   Specify that we inherit our sample time from the driving block.
*/
static void mdlInitializeSampleTimes (SimStruct *S)
{
    ssSetSampleTime (S, 0, INHERITED_SAMPLE_TIME) ;
    ssSetOffsetTime (S, 0, 0.0) ;
}

/* Function: mdlOutputs
* Abstract:
*    $y = 2*u$ 
*/
static void mdlOutputs (SimStruct *S, int_T tid)
{
    int_T i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs (S, 0) ;
    real_T *y = ssGetOutputPortRealSignal (S, 0) ;
    int_T width = ssGetOutputPortWidth (S, 0) ;
    for (i=0; i<width; i++) {
        *y++ = 2.0 * (*uPtrs[i]) ;
    }
}

```

```

    }
}
/* Function: mdlTerminate
 * Abstract:
 *   No termination needed, but we are required to have this routine.
 */
static void mdlTerminate (SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "Simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif

```

读者写这个源文件时，无需自己来全部输入，比较好的建议是使用 Simulink 提供的模板文件 `sfuntmpl.c`。它的使用方法和 M 文件 S-函数模板使用是一样的，读者只需把各个 S-函数方法的代码放到相应的位置，这样那些比较固定的语句，就不需要读者自己输入了。对于这个例子，还有一个更简便的方法。其实这里的 `doubleinput.c` 是从 `matlabroot/Simulink/src/` 目录下的 `timestwo.c` 变化而来的。其具体变化是将 `timestwo.c` 中的第一个语句

```
#define S_FUNCTION_NAME timestwo
```

中的 `timestwo` 变为 `doubleinput`。这样做的目的有两个：

其一，因为 Simulink 还提供了一个名为 `timestwo.m` 的 M 文件 S-函数，我们知道在 S-Function 模块中 M 文件 S-函数和 C MEX S-函数的调用形式是一样的，尽管 MATLAB 提供了一个估值顺序来区分两个 `timestwo` 函数（C MEX S-函数优先执行），但仍然容易混淆，所以不如改变名称来避免这些。

其二，改变名称后，读者可以自己尝试如何使用 `mex` 命令来生成 MEX 文件。因为在 `matlabroot/toolbox/Simulink/blocks` 目录下已经存在了一个编译好的 `timestwo.dll` 文件。

基于上述的原因，就将 S-函数的名称改为 `doubleinput`，于是文件名也要相应的保存为 `doubleinput.c`。

下面就对 `doubleinput.c` 进行简要的说明：

(1) 定义和包含语句。例程首先通过两个定义语句指定了 S-函数的名称（`doubleinput`）以及该 S-函数属于 level 2 格式。在定义完这两项后，源文件还包含了头文件 `simstruct.h`，它提供了对 `SimStruct` 数据结构和 MATLAB 应用程序接口函数的访问。我们知道，M 文件 S-函数中的各个方法都是通过 `sys` 参量来返回结果的，这些结果由 Simulink 自动完成和不同模型元素的对应。而在 C MEX S-函数中，各个 S-函数方法通过对数据结构 `SimStruct` 进行存取来完成和 Simulink 交互，读者可以从 `doubleinput.c` 的各个 S-函数方法的参量定义中看出这一点。为了能访问这个数据结构，就必须包含头文件 `simstruct.h`。在 `doubleinput.c` 中，完成定

义和包含的语句为：

```
#define S_FUNCTION_NAME timestwo
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

(2) `mdlInitializeSizes` 定义了 `doubleinput` 中的有关信息。这些信息的定义，都通过一些由 Simulink 提供的接口函数，对数据结构 `SimStruct` 进行读写来进行。例如，

```
ssSetNumSFcnParams (S, 0);
```

就是将 S-函数的参数个数设置为零（就是需要在 S-Function 模块对话框里设置的参数，相当于 M 文件的额外参数）。`doubleinput.c` 包含了大多数接口函数的用途，读者在自己编写过程中，只要把参数值替换一下就可以了。

`mdlInitializeSizes` 定义的信息有：

(a) 零参数，这意味着 S-函数模块的对话框中，S-Function 模块对话框的参数编辑框必须为空。如果它包含任何参数，Simulink 会报告一个参数失配。这通过一个 if 语句将设置的参数个数和实际输入的参数个数进行比较来判断是否是参数匹配，如果失配，则把控制权返回给 Simulink，由 Simulink 报告一个错误信息。请看这个语句：

```
if (ssGetNumSFcnParams (S) != ssGetSFcnParamsCount (S)) {
    return; // 控制权返回给 Simulink，然后，Simulink 将会报告参数失配
}
```

(b) 函数具有一个输入端口和一个输出端口，并且定义输入端口和输出端口的宽度都属于动态可变。对于动态可变的缺省处理是使输入和输出的宽度相同。

完成这输入端口和输出端口个数的设置的接口函数分别是：`ssSetNumInputPorts` 和 `ssSetNumOutputPorts`。在例程中，首先判断这两个函数是否成功地设置了端口个数，如果没有就把控制权返回到 Simulink。否则，就通过 `ssSetInputPortWidth` 和 `ssSetOutputPortWidth` 这两个接口函数分别设置输入端口和输出端口的宽度，宽度值可以是一个具体的整数。在这里由于是动态可变的，所以设置语句分别是：

```
ssSetInputPortWidth (S, 0, DYNAMICALLY_SIZED);
```

和

```
ssSetInputPortDirectFeedThrough (S, 0, 1);
```

这里要提醒读者注意，C MEX S-函数中，端口的编号是从 0 开始的，而不是 MATLAB 数组中的 1 开始。因此，上面的语句中函数的第二个参量都是 0。

在本例中，由于输出信号直接是将输入信号乘上 2，所以输入端口是直接馈入的，为此要用 `ssSetInputPortDirectFeedThrough` 借口函数来设置：

```
ssSetInputPortDirectFeedThrough (S, 0, 1);
```

其中的 0 代表要设置的端口，1 表示具有直接馈入，0 则是没有直接馈入。从这里读者可以看



出，在 C MEX S-函数源文件里设定函数的输入输出端口和 M 文件 S-函数的不同，前者是区分端口，它允许用户先设置端口的个数，再来设定每个端口的宽度。

(c) S-函数只有一个采样时间。关于采样时间的设置，C MEX S-函数和 M 文件 S-函数有所不同。C MEX S-函数里，在 `mdlInitializeSizes` 里只能通过 `ssSetNumSampleTimes` 来指定采样时间的个数，而采样时间的实际值用户必须在 `mdlInitializeSampleTimes` 方法里指定。

(d) 代码被设置为是无异常 (exception free) 的。指定代码为无异常代码，可以加速 S-函数的执行。当指定这个选项时，一定要特别小心。一般而言，如果 S-函数不是正在和 MATLAB 交互作用，指定这个选项是安全的。更详细的描述，参见后面的小节。完成这个设置的语句是：

```
ssSetOptions (S, SS_OPTION_EXCEPTION_FREE_CODE);
```

### (3) `mdlInitializeSampleTimes` 方法：

例程通过两条语句分别设置 S-函数的采样时间和偏移时间量大小。它的采样时间从驱动模块继承而来，这意味着 S-函数在接收到输入的任何时候，都会运行。而偏移时间则被设置为 0。

### (4) `mdlOutput` 方法如下所示。

这个方法的作用是进行数值计算，`mdlOutput` 将输入信号值乘以 2，把结果放到输出信号中。值得注意的有如下几点：

(a) 访问输入信号的方法是，使用 `ssGetInputPortRealSignalPtrs` 接口函数获得一个指向 `SimStruct` 中某端口的数据存储空间区域的指针，并把它赋给一个指针向量。

```
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs (S,0);
```

其中，`uPtrs` 是一个指针向量，它的数据类型是 `InputRealPtrsType`，这是 Simulink 自定义的类型，必须使用下面命令来访问该端口向量的第 *i* 个元素：

```
*uPtrs[i]
```

(b) 访问输出信号的方法和访问输入信号的方法类似，使用的语句是

```
real_T *y = ssGetOutputPortRealSignal (S,0);
```

这返回了模块输出的连续信号。

(c) 最后要注意代码中的循环语句。这个 S-函数可以处理向量输入信号。但 C 语言是不能像 MATLAB 那样直接进行向量运算，它是以标量计算为基础，所以要用一个循环次数为信号宽度的 For 循环来处理该端口的所有信号元素。为此，必须首先获得输入和输出端口的宽度，`ssGetOutputPortWidth` 函数可以获得输出端口的宽度。

### (5) `mdlTerminate`：

这是一种强制性的 S-函数方法。`timestwoS`-函数不需要实现任何的结束操作，因此这个方法是空的。

(6) 在 S-函数代码的末尾，指定和这个代码关联的代码是 Simulink 还是 Real-Time workshop：

```

#ifdef MATLAB_MEX_FILE
#include "Simulink.c"
#else
#include "cg_sfun.h"
#endif

```

### 9.3.3 建立更复杂的 C MEX S-函数

简单的 C MEX S-函数只包含一些必须的 S-函数方法，编写时，可以使用模板 `sfuntmpl.c` 来简化编写过程。但如果读者想建立更复杂的 C MEX S-函数，那就必须使用更复杂的模板文件 `sfuntmpl.c`，`sfuntmpl.doc` 里则有所有可以使用的方法的描述（在 `Simulink/src` 目录里）。

#### 1. S-函数顶端要求的语句

为了 S-函数正常的运行，每一个要访问数据结构 `SimStruct` 的 S-函数源代码块必须包含下面的定义和包含语句。

```

#define S_FUNCTION_NAME your_sfunction_name_here
#define SFUNCTION_LEVEL 2
#include "simstruc.h"

```

其中，`your_sfunction_name_here` 是 S-函数的名称。这些语句使得 S-函数可以访问数据结构 `SimStruct`，它包含了指向仿真中用到的数据的指针。这个 `incude` 代码同样定义了了在 `SimStruct` 中存储和获取数据的接口函数。

所谓的 level 1 格式 S-函数是在 Simulink 1.3 到 2.1 建立的，尽管它们和 Simulink 3.0 兼容，但建议最好是在 level 2 格式下重写 S-函数。

当 S-函数被编译为 MEX 文件时，`matlabroot/Simulink/include/simstruc.h` 包含的头文件如表 9-5 所示。

表 9-5 被编译为 MEX 文件时 `simstruc.h` 包含的头文件

头 文 件	说 明
<code>matlabroot/ extern/ include/ tmwtypes.h</code>	通用的数据类型，例如， <code>real_T</code>
<code>matlabroot/ extern/include/ mex.h</code>	MATLAB MEX 文件 API 方法
<code>matlabroot/ extern/ include/ matrix.h</code>	MATLAB MEX 文件 API 方法

当 S-函数被编译为和 RTW 一起工作时，`Simstrc.h` 包含的头文件有（表 9-6）。

表 9-6 应用于 RTW 时所包含的头文件

头 文 件	说 明
<code>matlabroot/extern/include/tmwtypes.h</code>	通用的数据类型，例如， <code>real_T</code>
<code>matlabroot/rtw/c/libsrc/rt_matrix.h</code>	MATLAB MEX 文件 API 方法

#### 2. S-函数低端规定的代码

在 S-函数的低端必须具备的代码有：

```

#ifdef MATLAB_MEX_FILE /* Is this being compiled as MEX-file? */
#include "Simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration func */
#endif

```

这些语句为用户选择合适的代码。如果 S-函数被编译为 MEX 文件，Simulink.c 文件被包含，如果这个文件是用于和 RTW 结合生成单机或者实时可执行代码时，cg\_sfun.h 文件被包含。

### 3. 条件编译 S-函数

S-函数可以被编译成由下面三种定义表示的三种模式之一：

- (1) MATLAB\_MEX\_FILE——表示 S-函数被编译为供 Simulink 使用的 MEX 文件。
- (2) RT——表示 S-函数被建立为使用固定步长解法器的实时应用的 RTW 实时代码。
- (3) NRT——表示 S-函数被建立为使用可变步长解法器的非实时应用的 RTW 生成代码。

### 4. 错误控制

当使用 S-函数时，正确地处理意料之外的事件十分重要，例如遇到无效的参数值。在上面的 timestwo 例子中，没有必要使用明显的错误控制。唯一实现错误控制的语句就是 mdlInitializeSizes 方法中前面出现的 return 语句。那里，处理了一个参数失配的情况，即使使用时在对话框输入参数。如果 S-函数具有需要进行有效性验证的参数，可以使用下面的技术来报告遇到的错误：

```

ssSetErrorStatus (S,"error encountered due to ...");
return;

```

注意 ssSetErrorStatus 的第二个参量必须是永久内存变量，它不能是过程中的局部变量。例如，下面的代码会导致无法预计的错误：

```

mdlOutputs ( )
{
char msg[256]; {ILLEGAL: to fix use "static char msg[256];"}
sprintf ( msg,"Error due to %s", string );
ssSetErrorStatus (S,msg);
return;
}

```

ssSetErrorStatus 错误控制方法是 mexErrMsgTxt 函数的推荐替代方法。函数 mexErrMsgTxt 使用异常处理来立即终止 S-函数的执行，并且返回控制给 Simulink。为了在 S-函数里支持异常处理，Simulink 必须在每个 S-函数调用之前安装异常处理器，这会增加仿真的花销。

### 5. 无异常代码

用户可以通过确信自己的 S-函数只包含无异常代码，来避免这种代价。无异常代码指没

有长跳转的代码，有潜在长跳转的代码都不是无异常代码。例如，`mexErrMsgTxt` 函数调用时，就会产生一个异常结束 S-函数的执行。使用 `mxMalloc` 将会在发生内存分配错误事件时产生无法预计的结果，因为 `mxMalloc` 存在长跳转。如果必须进行内存分配，请直接使用 `stdlib.h` 的 `calloc` 方法并且自己实现错误处理。

如果用户确信函数不会调用 `mexErrMsgTxt` 或者其他产生异常的 API 方法，那么就使用 `SS_OPTION_EXCEPTION_FREE_CODE` S-函数选项将代码设定为无异常。这通过 `mdlInitializeSizes` 函数里的下列命令来实现。

```
ssSetOptions (S, SS_OPTION_EXCEPTION_FREE_CODE);
```

设置这个选项将会使 Simulink 避开通常在每个 S-函数执行前进行的异常处理安装，这就可以提高 S-函数的性能。但使用这个选项一定要极为仔细，要确保代码是无异常代码。如果在这个选项设置后，S-函数产生了一个异常代码，那么就会发生无法预计的错误。

所有的 `mex*` 方法都存在长跳转的可能性，此外，几个 `mx*` 方法也具有长跳转的可能。为了避免产生问题，用户只能使用获得指针和决定参数大小的 API 方法。例如，下面的方法从不会产生异常：`mxGetPr`，`mxGetData`，`mxGetNumberOfDimensions`，`mxGetM`，`mxGetN`，`mxGetNumberOfElements`。

运行时方法 (*run-time routines*) 的代码也能产生异常，运行时方法指 Simulink 在仿真循环中调用的 S-函数方法。运行时方法包括：

- (1) `mdlGetTimeOfNextVarHit`
- (2) `mdlOutputs`
- (3) `mdlUpdate`
- (4) `mdlDerivatives`

如果 S-函数里所有的运行时方法都是无异常的，那么就可以在函数中使用这个选项：

```
ssSetOptions (S, SS_OPTION_RUNTIME_EXCEPTION_FREE_CODE);
```

而函数中的其他方法则无需是无异常的。

## 9.4 建立 C++ S-函数

建立 C++ S-函数是 Simulink4.0 的新增功能，它的过程和建立 C S-函数的过程极为相似，在这一节主要讲述它们之间的区别。

将 C++ S-函数源文件建立为可执行的 S-函数的方法和 C S-函数的方法是相同的，也是在 MATLAB 命令窗口输入：

```
>> mex sfun_counter_cpp.cpp % 将 mex sfun_counter_cpp.cpp 建立为同名的 S-函数。
```

只不过要注意，源文件的扩展名必须是 `.cpp`，这样使编译器把文件里的代码作为 C++ 代码来对待。

### 9.4.1 源文件格式

S-函数的 C++ 源文件的格式和用 C 编写的 S-函数的源文件几乎完全相同。主要的区别是 S-函数文件必须告诉 C++ 编译器一些信息，让它编译调用的 S-函数方法时，使用 C 调用规则。这是因为 Simulink 仿真引擎假定所调用的 S-函数方法遵循 C 调用规则。

要做到这一点，可以通过把 S-函数调用方法的 C++ 源文件，用一条 `extern "C"` 语句封装起来，`sfun_counter` 的 C++ 版本（`matlabroot/ simulink/ src/ sfun_counter_cpp.cpp`）演示了使用 `extern "C"` 语句，使得编译器产生与 Simulink 兼容的调用方法。`sfun_counter_cpp.cpp` 的代码如下：

```
/* File      : sfun_counter_cpp.cpp
 * Abstract:
 *
 *      Example of an C++ S-function which stores an C++ object in
 *      the pointers vector PWork.
 *
 *      Copyright 1990-2000 The MathWorks, Inc.
 *      $Revision: 1.1 $
 */
#include "iostream.h"
        声明 counter 类。
class counter {
    double x;
public:
    counter ( ) {
        x = 0.0;
    }
    double output (void) {
        x = x + 1.0;
        return x;
    }
};
#ifdef __cplusplus

extern "C" {
        // 使用 extern "C" 语句，让所有的函数采用 C 函数调用标准

#endif
        // defined within this scope
#define S_FUNCTION_LEVEL 2
```

```

#define S_FUNCTION_NAME  sfun_counter_cpp
                        // 定义 S-函数名为 sfun_counter_cpp

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions. */

#include "simstruc.h"
/*=====
 * S-function methods *
 *=====*/

/* Function: mdlInitializeSizes =====
 * Abstract:
 *   The sizes information is used by Simulink to determine the S-function
 *   block's characteristics (number of inputs, outputs, states, etc.) . */

static void mdlInitializeSizes (SimStruct *S)
{
    /* See sfuntmpl.doc for more details on the macros below */
    ssSetNumSFcnParams (S, 1) ; /* Number of expected parameters */
    if (ssGetNumSFcnParams (S) != ssGetSFcnParamsCount (S)) {
        /* Return if number of expected != number of actual parameters */
        return;
    }
    ssSetNumContStates (S, 0) ;
    ssSetNumDiscStates (S, 0) ;
    if (!ssSetNumInputPorts (S, 0)) return;

    if (!ssSetNumOutputPorts (S, 1)) return;
    ssSetOutputPortWidth (S, 0, 1) ;
    ssSetNumSampleTimes (S, 1) ;
    ssSetNumRWork (S, 0) ;
    ssSetNumIWork (S, 0) ;
    ssSetNumPWork (S, 1) ; // reserve element in the pointers vector
    ssSetNumModes (S, 0) ; // to store a C++ object
    ssSetNumNonsampledZCs (S, 0) ;
    ssSetOptions (S, 0) ;
}

/* Function: mdlInitializeSampleTimes =====
 * Abstract:

```

```
* This function is used to specify the sample time (s) for your
* S-function. You must register the same number of sample times as
* specified in ssSetNumSampleTimes.*/
```

```
static void mdlInitializeSampleTimes (SimStruct *S)
{
    ssSetSampleTime (S, 0, mxGetScalar (ssGetSFcnParam (S, 0))) ;
    ssSetOffsetTime (S, 0, 0.0) ;
}

#define MDL_START /* Change to #undef to remove function */
#if defined (MDL_START)
/* Function: mdlStart =====
* Abstract:
* This function is called once at start of model execution. If you
* have states that should be initialized once, this is the place
* to do it. */
static void mdlStart (SimStruct *S)
{
    ssGetPWork (S) [0] = (void *) new counter; // store new C++ object in the
                                                // pointers vector
}
#endif /* MDL_START */

/* Function: mdlOutputs =====
* Abstract:
* In this function, you compute the outputs of your S-function
* block. Generally outputs are placed in the output vector, ssGetY (S) .*/

static void mdlOutputs (SimStruct *S, int_T tid)
{
    counter *c = (counter *) ssGetPWork (S) [0]; // retrieve C++ object from
    real_T *y = ssGetOutputPortRealSignal (S, 0) ; // the pointers vector and use
    y[0] = c->output ( ) ; // member functions of the
                            // object
}

/* Function: mdlTerminate =====
* Abstract:
* In this function, you should perform any actions that are necessary
* at the termination of a simulation. For example, if memory was
* allocated in mdlStart, this is the place to free it. */

static void mdlTerminate (SimStruct *S)
```

```

{
    counter *c = (counter *) ssGetPWork (S) [0]; // retrieve and destroy C++
    delete c;                                // object in the termination
}                                              // function

/*=====
 * See sfuntmpl.doc for the optional S-function methods *
 *=====*/

/*=====
 * Required S-function trailer *
 *=====*/

#ifdef MATLAB_MEX_FILE /* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfunk.h" /* Code generation registration function */
#endif
#ifdef __cplusplus
}

    // extern "C" 作用域的结束

#endif

```

从上面的源文件，读者可以看到定义所有 S-函数调用方法的源代码，都被包含在 `extern "C"` 语句中，它如下所示。

```
extern "C" { 定义调用方法的源代码};
```

### 9.4.2 建立永久 C++对象

用户的 C++ S-函数调用方法也许要建立永久的 C++对象，也就是在退出方法的执行后也继续存在的对象。例如，一个调用方法可能要访问在它对前一次调用中建立的对象，或者是一个调用方法可能要访问另外一个方法所建立的对象。在自己的 S-函数里，建立永久 C++对象的步骤如下：

- (1) 建立一个指针工作向量，在方法调用之间保存永久对象的指针。其方法演示如下：

```

static void mdlInitializeSizes (SimStruct *S)
{
    ...
    ssSetNumPWork (S, 1);
    // 在 指针向量里保留一个元素来存储 C++对象。
    ...
}

```



其中，`ssSetNumPWork` 方法的作用是在数据结构 `SimStruct` 里设置指针工作向量的数目，它的第二个参量，就表示要设置指针工作向量的个数，在上面的代码里是 1。

(2) 为要设为永久的每个对象各自存储一个指针到指针工作向量里。例如，

```
static void mdlStart (SimStruct *S)
{
    ssGetPWork (S) [0] = (void *) new counter;
                                     // 在指针向量
}
                                     // 存储新的 C++ 对象
```

因为在初始化函数里，已经设置了一个指针工作向量，所以在这里，就可以用 `ssGetPWork (S)` 来获得这个指针数组，因为 C++ 数组是从 0 开始计数的，所有数组的下标就是 0。

(3) 在后面的方法调用中从指针向量获得原来保存的指针，以访问对象。例如，

```
static void mdlOutputs (SimStruct *S, int_T tid)
{
    counter *c = (counter *) ssGetPWork (S) [0];
                                     // 从指针向量获得 C++ 对象，并
                                     // 使用对象的成员函数。

    real_T *y = ssGetOutputPortRealSignal (S,0);
    y[0] = c->output ();
}
}
```

(4) 在仿真结束时，销毁对象。例如，

```
static void mdlTerminate (SimStruct *S)
{
    counter *c = (counter *) ssGetPWork (S) [0]; // 在 mdlTerminate 函数里获得，
                                     // 并且销毁 C++ 对象

    delete c;
}
}
```

## 第十章 使用 Real-Time Workshop

### 10.1 Real-Time Workshop 综述

#### 10.1.1 Real-Time Workshop 能做什么

在进入具体的学习之前,我想读者应该先弄清楚 Real-Time Workshop(以后简称为 RTW)能作些什么,对自己的工作是否有帮助。

简而言之,RTW 是和 MATLAB、Simulink 一起使用的一个工具,它可以直接从 Simulink 模型生成代码并且自动建立可以在不同环境下运行的程序,这些环境包括实时系统和单机仿真。单机仿真的概念是指,用户可以不用打开 MATLAB 和 Simulink 模型图表就可以运行仿真,因为 RTW 根据模型建立了一个可执行的程序。

RTW 提供的另外一个很有诱惑力的功能是,通过它,用户可以在远程处理器实时的运行 Simulink 模型,并且它所生成的单击仿真程序同样具备在外部计算机运行的能力。

RTW 能够应用的场合是十分广泛的,下面列举了它的几个例子:

(1) 实时控制——读者可以使用 MATLAB 和 Simulink 设计控制系统,并且从建立的模块图表模型生成代码。读者可以编译它们和载入它们到目标硬件。

(2) 实时信号处理——读者可以使用 MATLAB 和 Simulink 设计信号处理算法,同样可以从模型生成代码,编译和载入它们到目标硬件。

(3) 硬件环路仿真——读者可以建立模仿实际生活测量,系统动力和激励信号的 Simulink 模型。从模型生成的代码可以被定位到特殊用途的硬件(如 DSP),它提供了物理系统的实时表示。

(4) 交互的实时参数调整——读者使用 Simulink 作为建立的实时程序的前端,就可以利用 Simulink 的图形界面,在程序执行时改变参数。

(5) 高速的单机仿真。

(6) 生成可插入到其他仿真程序的便携 C 代码。

无论是何种应用,代码生成都是其中必需的过程。RTW 生成的代码在缺省情况下是高度优化和完全注释的 C 代码。从任何 Simulink 模型都可以生成代码,它们包括:线性、非线性;连续,离散以及混合模型。

所有的 Simulink 模块都自动地转化为代码,除了 MATLAB function 模块和调用 M 文件 S-函数的 S-function 模块。如果想把它们和 RTW 一起使用,就需要重写这些模块为 C MEX S-函数。

RTW 包含一系列的目标文件,并由目标语言编译器(TLC)编译生成 ANSI C 代码。目标文件是 ASCII 文本文件,描述如何将 Simulink 模型转化为代码。对于高级用户,目标文件

使得它能够自定义生成代码。

读者还可以把 C MEX S-函数和生成的代码一起合并到可执行程序。同样还可以为自己的 C MEX S-函数编写一个目标文件来内嵌 S-函数，这样就可以通过减少对 S-函数的调用来提高性能。

RTW 可以输出的类型有：

(1) C 代码——为 Simulink 模型生成包含系统方程和初始化函数的代码，读者可以在非实时环境或者实时环境使用这些代码。

(2) Ada 代码——从 SIMULINK 模型生成 Ada 代码，这要求用户安装 Real-Time Workshop Ada Coder。

(3) 实时程序——将代码转换为适合指定实时硬件运行的实时程序。对应的代码被设计为和一个外部时钟源相连接，并且以一个固定的由用户设定的采样速率运行。

(4) 高性能的单机仿真——将生成代码和普通实时系统目标文件一起使用，为单机仿真生成可执行程序。在仿真结束时，可执行文件产生一个 *model.mat* 文件，它里面包含了 Simulink 保存仿真数据的 MATLAB 变量，这个文件可以在 MATLAB 里进行分析。

在大致了解 RTW 的主要用途之后，读者如果对 RTW 的使用有兴趣，那么就请继续阅读本章的余下内容。RTW 的内容是相当多的，本章只介绍它的一些比较基本的使用，关于更详细的说明，读者可以参阅 MATLAB 的帮助文档。

### 10.1.2 使用前的准备工作

在读者确信 RTW 对自己的工作有所帮助之后，那么请您在进入学习之前，先做一些准备工作，这主要是一些软件的安装和选项的简单设置。

首先，读者当然要在所使用的机器上安装好 RTW，这一点使用 MATLAB 安装向导是很容易办到的。

接着，读者要安装一些第三方编译器，这些编译器是在读者使用 RTW 生成可执行程序时，供 RTW 访问调用的。为此，还要进行一定的设置。RTW 支持的几种第三方编译器如下。

Watcom——支持的版本有：10.6，11.0；

Borland——支持的版本有：5.0，5.2，5.3；

Microsoft Visual C/C++——支持的版本有：4.2，5.0，6.0。

读者可以选择其中的一种安装，并且按照下面的描述进行设置。首先要设置的是环境变量，它要求正确的指向编译器所在的位置。

(1) Watcom。定义 WATCOM 环境变量，在 Windows95 或者 98 下，读者可以把下面的命令添加到 autoexec.bat 文件，这个文件在 C:\Windows 目录下。

```
set WATCOM= <Watcom 编译器的安装路径>
```

例如，set WATCOM= D:\Watcom。

可以通过在 dos 提示符下输入

```
set WATCOM
```

来检查 WATCOM 环境变量是否定义，并且正确的指向 Watcom 编译器所在的目录。这个命令将返回所选择的目录。

在 Windows NT 下，可以在控制面板选择 System，进入 Environment 页，然后定义

WATCOM 变量，并且输入编译器的路径。

(2) Borland。为 Borland 编译器设置环境变量的过程和 Watcom 编译器基本上一样，唯一的区别在于环境变量名是 BORLAND。

(3) Microsoft Visual C/C++。对于 Microsoft Visual C/C++ 编译器，则要定义环境变量 MSDevDir 为：

MSDevDir=<编译器的路径> for Visual C/C++ 4.2

MSDevDir=<编译器的路径>\SharedIDE for Visual C/C++ 5.0

MSDevDir=<编译器的路径>\Common\MSDev98 for Visual C/C++ 6.0

例如，set msdevdir=c:\program files\microsoft visual studio\common\msdev98。

在设置过程中，如果遇到“out-of-environment space”这样的错误信息，读者可以用鼠标右键单击产生这个问题的程序（例如，autoexec.bat），并且从弹出的菜单中选择 Properties 命令，从打开的对话框中选择 Memory 页面，并把 Initial Environment 属性设为所能允许的最大值，然后按 Apply 按钮应用上面的改动。

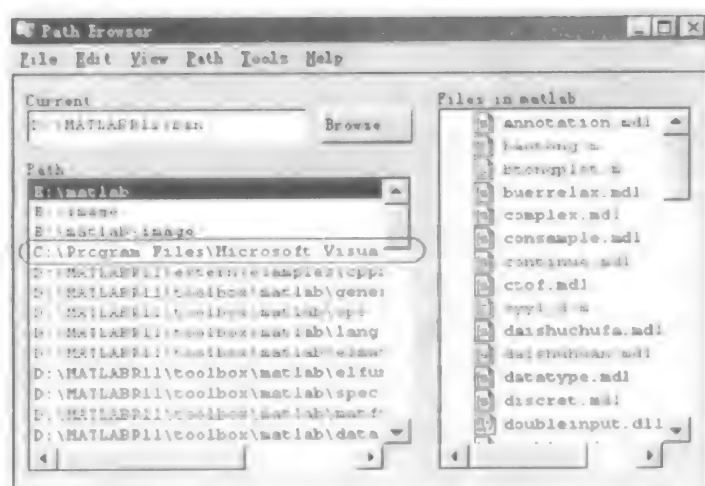


图 10-1 添加 make 到搜索路径

其次，还要进行的一项设置是修改 MATLAB 的搜索路径。因为 RTW 在建立程序时，会调用第三方编译器的 make 程序，为了让 RTW 能够找到 make 程序，读者要把 make 程序所在的目录添加到 MATLAB 的搜索路径中。例如，假如使用的第三方编译器是 Microsoft Visual C/C++ 6.0，那么添加路径后的 MATLAB 路径浏览器如图 10-1 所示。

### 10.1.3 RTW 中的基本概念

在一步一步地学习使用 RTW 的一个个例子之前，读者有必要先了解 RTW 中的一些基本概念。这些概念包括：

- (1) 普通实时；
- (2) 目标定位；
- (3) 目标语言编译文件；
- (4) 建立过程；
- (5) 模板 makefile；

- (6) 配置模板文件;
- (7) 指定模型参数。

#### 1. 普通实时

RTW 提供了普通的实时发展对象, 所谓普通实时是:

- (1) 一个在单任务或者多任务模式下仿真固定步长模型环境;
- (2) 实现代码验证的一种方法。

#### 2. 目标定位

使用 RTW, 读者必须决定放置生成代码运行的环境, 包括硬件或者操作系统, 这称为目标定位, 环境本身就成为目标 (target)。而宿主 (host) 就是用户运行 MATLAB、Simulink 和 RTW 的系统。使用 RTW, 可以生成代码以及能够在目标系统中的可执行程序。生成特定目标的代码的过程由系统目标文件、模板 makefile 文件和 make 命令来控制。系统目标文件和模板 makefile 文件定义了目标的类型, 它们必须在仿真参数对话框的 Real-Time Workshop 页被设定。

#### 3. 目标语言编译文件

目标语言编译文件 (Target Language Compiler files 或 TLC), 是供目标编译器编译和执行的文件, 它描述了如何将 Simulink 模型翻译为用户所设定的目标代码。RTW 使用 TLC 文件来把 Simulink 模型翻译成代码, 而系统目标文件是 TLC 程序生成可执行程序的进入点。

#### 4. 建立过程

RTW 建立过程由 make\_rtw 来控制, 这个程序在用户点击 Real-Time Workshop 页上的 Build 按钮 (见后面小节) 后被调用。首先, make\_rtw 编译模块图表并且生成一个 model.rtw 文件。然后, make\_rtw 调用目标语言编译器来生成代码, 但系统目标文件必须先设定。紧接着, make\_rtw 从 Real-Time Workshop 页上指定的模板 makefile 生成一个 makefile, 名为 model.mk。最后, 如果运行的 host 和模板 makefile 的 HOST 宏匹配, 那么就调用 make 程序? 从生成代码建立程序。

#### 5. 模板 makefiles

RTW 使用模板 makefile 从生成代码建立可执行程序。按照惯例, 一个模板 makefile 有一个 .tmf 的后缀名, 还有一个和所应有目标相对应的名称。例如, grt\_unix.tmf 是用于 UNIX 的普通实时模板 makefile, 这个模板文件应用的目标是 grt\_unix。

从模板 makefile 生成的 makefile 逐行地复制模板 makefile 的每一行, 并且把记号扩展到 makefile。生成的 makefile 名为 model.mk, 它被传给 make 命令, make 再根据一些文件来生成可执行程序。

#### 6. 模板 makefile 配置

读者可以通过修改模板 makefile 来配置建立过程。完成的过程可以是, 把模板文件复制到你的当前工作目录, 并且编辑它。或者可以通过在 make\_rtw 命令后面添加选项来配置模板 makefile。例如, 为 grt\_unix.tmf 打开调试符, 读者可以在仿真参数对话框设定建立命令为:

make\_rtw OPT\_OPTS=-g % OPT\_OPTS=-g 就是添加的选项设置

#### 7. 设定仿真参数

读者知道, 修改仿真参数可以控制仿真的行为, 例如起始和终止时间, 关于它们的设置, 本书的前面章节已经介绍得很清楚, 这里就不再赘述。请读者记住, 仿真参数直接影响到代



在介绍如何生成代码之前，让我们来粗略地审视一下 f14。读者可以看到，f14 模型有 4 个比较熟悉的模块：一个信号发生模块和 3 个 Scope 模块。其中，信号发生模块用来模拟飞行员的操作输入，它的波形由一个 Scope 模块来跟踪，而剩余的 Scope 模块则用来跟踪系统的输出——攻击的飞行角度和飞行员所承受的重力，使用 Scope 模块显示波形是 Simulink 里跟踪模型行为的常用方法。至于模型中的其他模块（系统），读者可以自己一层层的打开来查看。它们涉及到 f14 模型的设计和内部工作原理，读者可能会对这些很迷惑，但也不必担心。这个示例的目的只是让读者知道如何从 Simulink 模型生成普通的实时代码，而生成代码所用到的操作都是和模型的内容和复杂性无关的。

需要强调的一点是，读者也可以使用别的 Simulink 模型来生成实时代码。

## 10.2.2 生成实时代码

### 1. 设置程序的参数

生成实时代码首先要进行的操作是设置一些参数。在打开 f14 模型之后，请读者从下拉菜单 **simulation** 里选择 **parameters** 命令，打开仿真参数对话框，这个对话框的大部分页在本书的前面章节已经详细讲过，只有 **real-time workshop** 页涉及不多，图 10-3 显示了这个页的样子，该页包含的操作将在本章详细介绍。

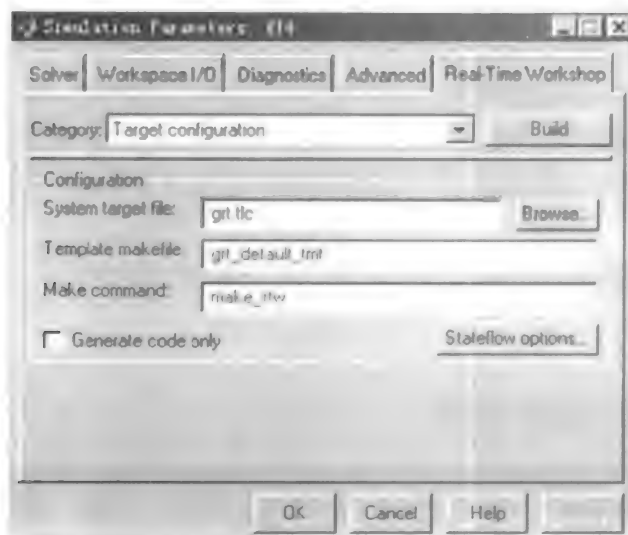


图 10-3 real-time workshop 页

**Real-Time workshop** 页是用户设置参数选项，选择模板制造文件（makefile），产生代码和建立程序的地方。在对话框里初始显示的缺省参数值不一定会和用来生成可实时执行的代码所要求的参数匹配。因此，Simulink 要求用户在自己的模型使用 **Real-Time Workshop** 时，改变缺省参数来和模型的仿真参数匹配。要为 f14 模型进行配置，请读者按照下面所述来进行：

（1）在 **Solver** 页，设置 **Solver options Type** 参数为 **Fixed-step**，并且选择 **ode5 (Dormand-Prince)** 解法器。

（2）设置 **Fixed Step Size** 参数为 0.05。

设置完参数之后，就可以进入建立普通实时程序的环节了。

☞ 注意，打开 Real-Time Workshop 页的方式还有从 Tools 下拉菜单选择 Real-Time workshop 的 Options 命令。

## 2. 建立程序

要建立程序，请读者在仿真参数对话框里选择 Real-Time Workshop 页，然后按 Build 按钮就可以从 f14 模型生成 C 代码。build 命令调用一个系统目标文件 grt.tlc，它使用指定的模板文件 rt\_default\_tmf，来生成 makefile。Real-Time Workshop 再使用生成的 makefile 来建立程序。

当对话框的 build 按钮被按下之后，Real-Time Workshop 按下面的顺序调用 make 命令：

(1) 编译模块图生成 model.rtw 文件（在本例中，是 f14.rtw）。

(2) 调用目标语言调用器，依次编译 TLC 程序，从 grt.tlc 开始和处理 model.rtw 来生成代码。

(3) 从模板 makefile 文件（例如，grt\_vc.tmf）建立一个名为 model.mk 的 makefile。

(4) 如果 Simulink 在和模板 makefile 文件所指定的相同宿主机上运行，那么实时程序就生成好了，否则，在生成代码后处理就中断，除非用户在模板 makefile 定义和目标相匹配的宿主机。

在模板 makefile 文件里定义的变量 HOST 用以识别用户的目标系统。HOST 的取值有三个选项：PC、UNIX 和 ANY。其中 ANY 的作用是让用户的目标系统不是所运行的这个（例如目标系统是 DSP 或者微处理器等等）。

一旦 build 命令被执行，Real-Time Workshop 缺省情况就生成下列文件：

(1) f14.c——单机 C 代码。

(2) f14.h——包含状态变量信息的包含头文件。

(3) f14\_export.h——包含输出信号和参数的包含头文件。

(4) f14.reg——一个包含头文件，它包含了完成在生成代码你对数据结构进行初始化的模型注册函数信息。

(5) f14.prm——一个包含模型 f14 中使用的参数的有关信息的包含头文件。

(6) f14 (UNIX) 或者 f14.exe (PC 机上)——普通的实时可执行代码。

下面的执行结果就是在 PC 机上（操作系统为 Windows）执行 build 命令后，MATLAB 命令窗口显示的信息。当然显示在 MATLAB 命令窗口的信息和你的仿真参数设置有关（请见 RTW 用户界面一节）。

```
### Starting RealTime Workshop build procedure for model: f14
### Invoking Target Language Compiler on f14.rtw
### f14.mk which is generated from D:\MATLABR11\rtw\c\grt\grt_vc.tmf is up to date
### Building f14: f14.bat
### Successful completion of RealTime Workshop build procedure for model: f14
```

## 3. 自定义 build 的过程

Simulink 还允许用户可以选中 Inline parameters 检查框来选中内置参数。这会导致 Real-Time Workshop 用模型参数数值的取值替换变量名，来建立普通的实时程序。它同样指 Simulink 传递常数采样时间，这样可以把常数运算放入启动代码，改善代码的性能。Simulink 同样允许用户自定义模板 makefile 和 make 命令；这些选项将会在本章的后面讨论。



单击 **Generate code only** 告诉 Real-Time Workshop 只产生代码而不编译, 意味着目标和可执行文件没有生成。选择 **Retain .rtw file** 将使 Real-Time Workshop 保存 **model.rtw** 文件。缺省情况下, 这个文件在 **build** 过程中被删除。

#### 4. 数据记录

这个例子使用 Real-Time Workshop 的 MAT-file 数据记录功能来保存系统状态、输出以及仿真时间。为此, 请在 **Workspace I/O** 页选中 **Time**、**States** 和 **Outputs** 选项使 Real-Time Workshop 记录仿真时间、任何离散或者连续状态和任何根输出端口模块。图 10-4 是 **Workspace I/O** 页的样子。

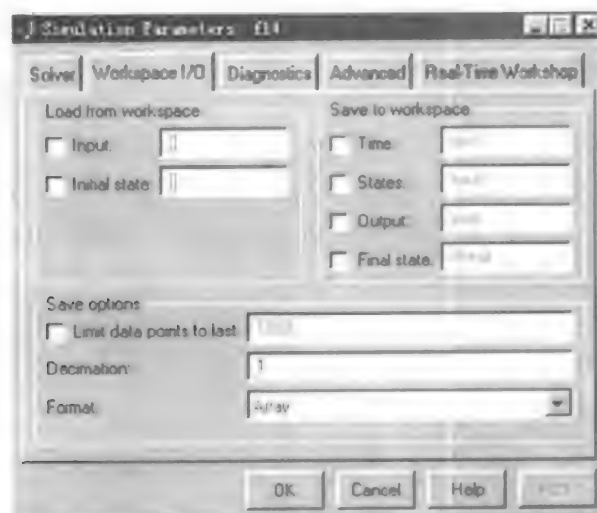


图 10-4 Workspace I/O 页

例如, 可以选中 **Time and Outputs** 选项来设置数据记录变量, 此外, Real-Time Workshop 也会把所有设置了保存数据到工作空间属性的所有 **scope** 模块的输入数据保存到一个 MATLAB MAT 文件。于是, 用户就可以载入数据到 MATLAB 作为分析。

具体操作是选中 **scope** 模块的特性设置对话框的数据历史 (**Data history**) 页的 **Save data to workspace** (保存数据到工作空间) 选项, 然后再在下面的编辑框里输入存储数据的变量名称。图 10-5 所示的是 **Stick Input** 模块的属性对话框, 读者可以照此来设置其他的两个 **scope** 模块。

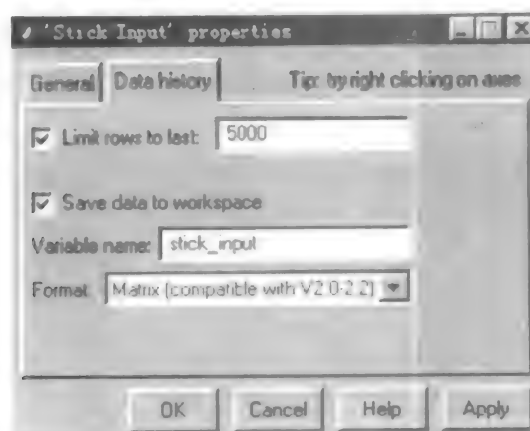


图 10-5 设置 scope 模块

如此设置之后，就可以重新用 **build** 命令来生成可执行代码，然后请读者在命令窗口输入如下的命令

```
>> !f14           % 执行生成可执行的程序

**starting the model**
** created f14.mat **

>> clear         % 清除以前在 MATLAB 工作空间已有的变量，以免混淆。
>> load f14
>> who
Your variables are:
rt_Angle_of_attack      rt_stick_input
rt_pilot_G_force
```

f14.mat 文件里包含了可执行程序执行之后生成的几个输出数据变量，读者可以发现变量的名称和设置的变量不一样，它们都多了一个前缀 **rt\_**，“**rt\_**”是 **real time** 的简写，在后面读者会知道这个前缀是可以设置的。

注意，使实时代码具有数据记录能力，可以通过设置 **Workspace I/O** 页的保存数据到工作空间属性，也可以通过 **scope** 模块的属性对话框来设置。这两种方式是互不影响的。例如，读者可以只用 **Workspace I/O** 页来设置，例如选择时间和输出（请详见第四章的保存数据到 **MATLAB** 工作空间），运行据此生成的代码就会把输出端口 1 和 2 的输出以及时间值保存到 **mat** 文件。

## 5. 实时运行的接口

建立实时程序除了要根据模型生成代码之外，还需要一系列的源文件。这些源文件包括：

- (1) 主程序；
- (2) 驱动模块执行的代码；
- (3) 实现积分算法的代码；
- (4) 实现数据记录的代码；
- (5) 生成 **SimStruct** 数据结构的代码，**SimStruct** 用于管理模型的执行。

## 6. 设置模板 makefile

这个例子运用了两个不同的模板 **makefiles**：用于 **UNIX** 的 **grt\_unix.tmf**，用于 **PC** 平台的 **grt\_vc.tmf**、**grt\_watc.tmf** 和 **orgrt\_bc.tmf**（取决于所选择的编译器）。这些 **makefile** 模板自动处理设置好的 **makefiles** 来建立程序。用户可以把模板 **makefile** 复制到用户的当前目录，然后修改它，来修改建立可执行程序的过程。

这些文件的位置位于 **matlab/rtw/c/grt** 目录，读者可以用 **MATLAB** 的 **matlabroot** 命令决定自己所用系统的 **MATLAB** 路径。

关于 **UNIX** 模板 **makefile**，这里就忽略不讲，请使用 **UNIX** 环境的读者查阅 **RTW** 的帮助文档。主要来介绍面向 **PC** 的 **makefiles**。其中 **grt\_vc.tmf** 和 **grt\_msvc.tmf** 是为使用 **Microsoft Visual C/C++** 编译器的用户设计的。要使用这些模板 **makefiles** 的任何一个，用户必须确保指

向 Visual C/C++ 的环境变量已经被定义，并且 `nmake` 命令所在目录已经被添加到 MATLAB 的搜索路径中。即保证以下几点：

- (1) `MsDevDir` 环境变量已定义；
- (2) Visual C/C++ 已正确安装；
- (3) `nmake` 已添加到 MATLAB 的搜索路径中。

而模板 makefile——`grt_bc.tmf`，是为使用 Borland C/C++ 编译器而设计的，使用 `grt_bc.tmf` 也要保证：

- (1) `BORLAND` 环境变量已正确定义；
- (2) Borland 编译器被正确安装。

7. 普通实时模块

源模块由 makefile——`model.m` 来自动链接。这个例子中用到的模块有：

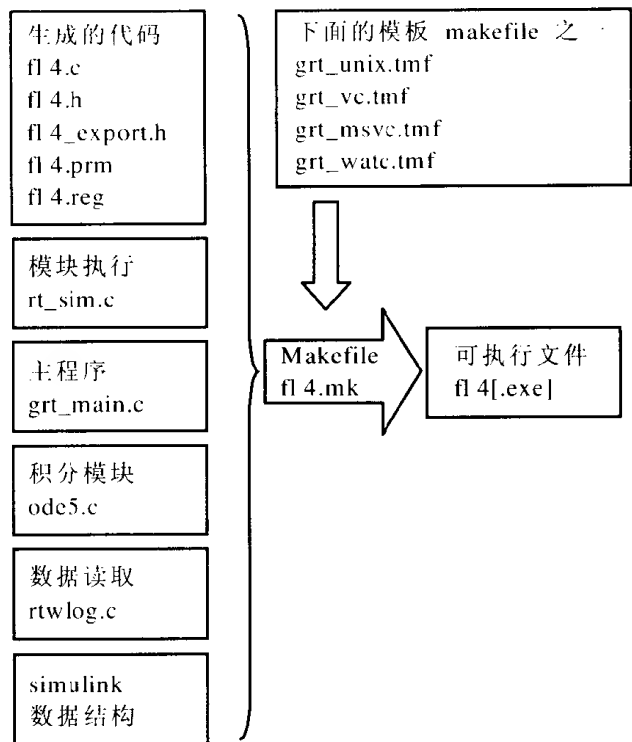


图 10-6 用于建立程序的源代码块

图 10-6 反映了用于建立普通实时程序的代码模块。

8. 依赖于绝对时间的模块

某些 Simulink 模块使用绝对时间值（即从仿真开始到仿真的当前时刻的时间，注意这是指仿真模型里的时间表示，而不是运行仿真所用的时间）来计算输出。如果用户正在设计一个有可能会一直运行的程序，那么用户不能使用对绝对时间有依赖性的模块。原因很简单，在时间的值达到了 `double` 精度数字所能表示的最大值时，问题就出现了。在这时，时间就不再增加，于是模块的输出就不再正确了。

表 10-1、10-2 和 10-3 分别列出了所有依赖绝对时间的模块。

表 10-1 依赖绝对时间的模块之一

连续模块	离散模块	非线性模块
Derivative Variable	Discrete-Time Integrator (在触发子系统)	Rate Limiter
Transport Delay		
Transport Delay		

表 10-2 依赖绝对时间的模块之二

接收器	
To Workspace (仅仅在保存的格式是带时间的结构时)	
Scope	
To File	

表 10-3 依赖绝对时间的模块之三

源模块	
Chirp Signal Pulse Generator	Discrete Pulse Generator Signal Generator
Clock Ramp	From Workspace Sine Wave
Digital Clock Repeating Sequence	From File Step.Code Validation

此外，在仿真参数对话框的 Workspace I/O 页选中 time 这个检查框来保存时间，也需要绝对时间。

10.2.3 代码验证

在完成了可执行程序的建立之后，f14 模型的单机仿真程序版本现在可以和 Simulink 模型进行性能比较了。前面讲过，可执行程序将数据 pilot G forces、aircraft angle of attack 和仿真时间，保存到 MAT 文件。读者可以通过设置 Scope 的 Save data to workspace 选项，把控制输入信号保存到 MATLAB 工作空间。这样，读者就可以用 MATLAB 命令来绘出 MAT 文件所保存数据的曲线图。在 f14 的 Simulink 版本和普通实时单击版本，模拟控制输入用的方波频率为 0.5 (rad/sec)，幅度为正负 1，均值为 0。

请读者先运行 Simulink 仿真模型，仿真的时间从 T=0 到 T=60。图 10-7 显示了 Simulink 模型仿真运行输出的结果。然后再运行 RTW 所建立的可执行程序，比较它产生的结果和 Simulink 模型输出的有什么不同。

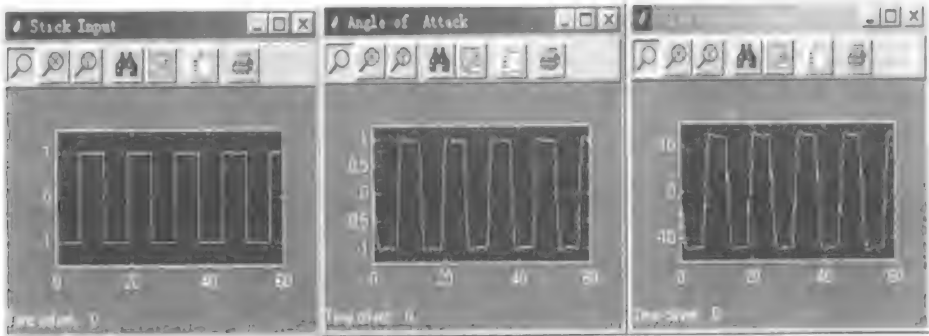


图 10-7 Simulink 模型得到的结果

### 1. 运行单击程序

下面从 MATLAB 命令窗口运行单机程序。

```
>> !f14
```

！在这里的作用是把跟在后面的命令传给操作系统，于是这个命令就是运行 f14 的单机程序版本（而不是 M 文件）。要获得单机程序的运行结果，可以载入 f14.mat 文件到工作空间。

```
>>clear % 先把 Simulink 模型运行后存储到 MATLAB 工作空间的变量清除。
```

```
>>load f14
```

再查看工作空间中的变量：

```
>> who
```

Your variables are:

```
rt_Angle_of_Attack      rt_tout
rt_Pilot_G_force        rt_yout
rt_Stick_Input
```

变量 rt\_tout、rt\_xout 和 drt\_yout 的产生是因为在 Workspace I/O 选中它们相应的检查框的原故，而变量 rt\_Pilot\_G\_force、rt\_Angle\_of\_attack 和 rt\_Stick\_input 这是使用模型 f14 里的 Scope 模块的属性页来保存的。

对使用 Scope 模块不熟悉的读者，可以查阅本章前面的内容。这些变量的命名取决于产生它们的 Simulink 模块。对所有的名称都会加上前缀 rt，表示实时数据。

于是就可以使用 MATLAB 命令来绘出 3 个工作空间变量相对于时间的曲线。

```
>> plot (rt_tout, rt_Stick_Input (:,2) );
>> figure;
>> plot (rt_tout, rt_Pilot_G_force (:,2) );
>> figure;
>> plot (rt_tout, rt_Angle_of_Attack (:,2) );
```

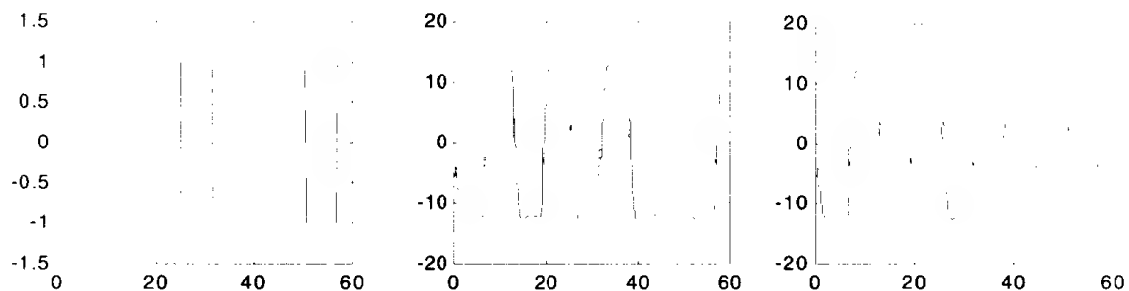


图 10-8 实时程序得到的仿真结果

图 10-8 演示了实时程序得到的仿真结果，其各变量的排列顺序和图 10-7 相对应。

## 2. Simulink 模型和普通实时程序结果的比较

读者可能已经注意到 Simulink 仿真和实时代码产生几乎相同的结果。现在就来比较 Simulink 模型的输出和 Real-Time Workshop 获得的仿真结果。请按下面的步骤来进行一个正确的比较：

(1) 首先，确信 Simulink 和 Real-Time Workshop 建立过程采用相同的积分机制（必须是固定步长），例如都是 ode5（Dormand-Prince），同时也要把固定步长设置为相同的数值（例如，0.05）。

(2) 设置 Scope 模块的 Save data to workspace 选项，以保存 f14 模型的输入和输出（例如，可以让攻击的飞行角 Scope 模块输出变量为 Angle\_of\_attack，飞行员所受重力 Scope 模块输出为 Pilot\_G\_Force，控制输入 Scope 模块输出为 Stick\_input）。

(3) 运行 f14 仿真，指用 Simulink 模型。

(4) 建立 f14 普通实时系统，运行它，并且载入 MAT-file——f14.dat，到 MATLAB 工作空间。现在，在 MATLAB 工作空间里，就有两种数据，一种由 Simulink 模型产生，而另一种则由 Real-Time Workshop 建立的可执行程序产生。这一点可以使用 who 命令来证实。

```
>> who
```

```
Your variables are:
```

```
Angle_of_attack rt_Stick_input
```

```
Pilot_G_force Stick_input
```

```
rt_Angle_of_attack
```

```
rt_Pilot_G_force
```

比较 Angle\_of\_attack 和 rt\_Angle\_of\_attack，得到：

```
>> max(abs(rt_Angle_of_attack-Angle_of_attack))
```

```
ans =
```

```
1.0e-015 *
```

```
0      0.5551
```

比较 Pilot\_G\_force to rt\_Pilot\_G\_force，得到：

```
>> max(abs(rt_Pilot_G_force-Pilot_G_force))
```

```
ans =
```

```
1.0e-012 *
```

0 0.1007

总的偏差范围在  $10^{-12}$  以内。这个小误差由许多因素导致，包括：

- (1) 不同的编译器优化；
- (2) 语句的顺序；
- (3) 运行库。

例如，计算  $\sin(2.0)$  就会因为所用的 C 代码库不同而不同。

如果要进一步的分析数据，用户可以把 To Workspace 模块加入到 Simulink 模型中，于是就可以访问模型中任何模块所产生的数据，并把它们保存到 MATLAB 工作空间变量中去。

## 10.3 代码生成和建立过程

Real-Time Workshop 的一个作用就是简化了建立应用程序的过程。Real-Time Workshop 的一个特性就是 automatic program building (自动程序建立)，它提供了一种标准方法，建立可在不同的目标环境下使用的实时程序。这种方法是单一的，也是受控的，用户可以对它进行自定义。

自动程序建立使用了 make 命令来控制程序的建立，它还使用一个 M 文件从一个可定制的模板来建立一个定制过的 makefile (被 make 命令引用的描述文件)。前面一节介绍了建立的过程和模板 makefile，在这一节将更为详细的讨论这些概念。包括的主题有：

- (1) 自动程序建立；
- (2) Real-Time Workshop 用户界面；
- (3) 配置生成的代码；
- (4) 定制模板 makefile；
- (5) 普通的实时模板 makefiles。

### 10.3.1 自动程序建立

Real-Time Workshop 自动完成从 Simulink 模型建立一个单机程序的任务。前面讲过，当 build 按钮一旦被按下，make 命令被调用，从 make\_rtw 开始。调用的过程包括三个主要步骤，它们受一个 M 文件 make\_rtw.m 控制。

- (1) 生成模型代码；
- (2) 生成定制一个指定建立过程的 makefile；
- (3) 调用具有定制过的 makefile 的 make 命令。

图 10-9 反映了建立的过程。当 Build 按钮被单击之后，上面的这些步骤就被自动完成。

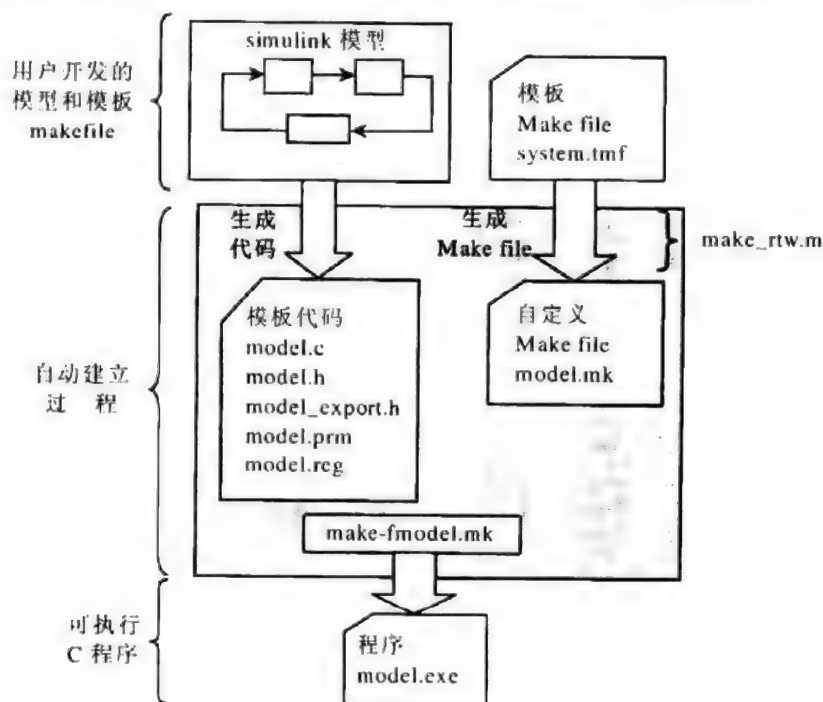


图 10-9 建立过程

### 10.3.2 Real-Time Workshop 用户界面

为了方便用户使用 RTW, Simulink 提供了一个用户界面来控制 RTW 的执行, 它包含在仿真参数对话框里, 其中最主要的就是 Real-Time Workshop 页。Real-Time Workshop 页只对 Real-Time Workshop 有效, 而其他的页则对 Simulink 仿真和 Real-Time Workshop 都有效。也就是说仿真参数对话框里的所有页面都影响 Real-Time Workshop 实时代码的生成。使用 Real-Time Workshop 时, Solver 页上的解法器 (Solver) 选项必须设置为固定步长的解法器 (fixed-step solver)。Workspace I/O 页是设定数据保存选项的, Diagnostics 页是指定产生错误警告的选项, 这些页在本书的第五章有详细的介绍, 这里不再赘述。

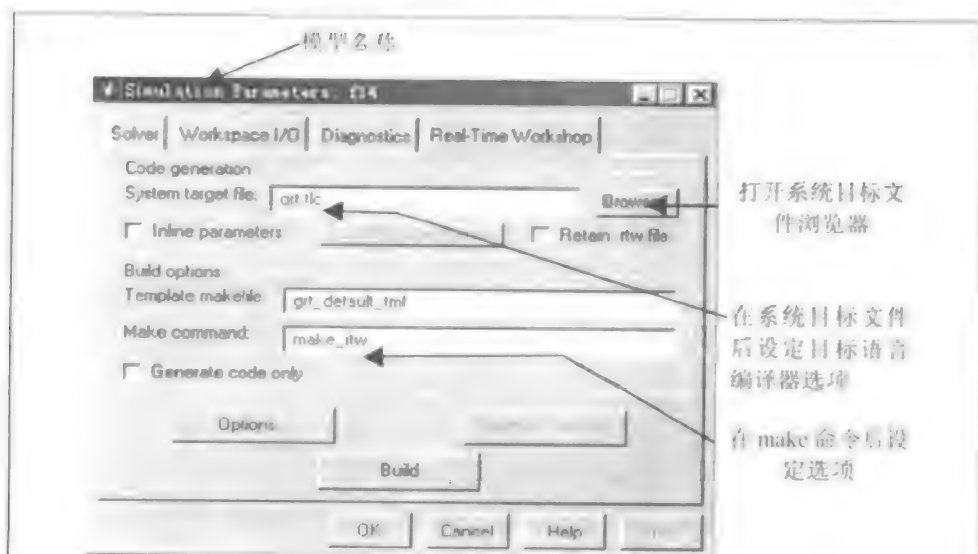


图 10-10 Real-Time Workshop 页



关于 Real-Time Workshop 页，在前面一节，我们仅仅用到了其中的 build 命令，其他的编辑框和选项检查框都没有涉及，下面就来具体介绍它们。请读者对照 Real-Time Workshop 页来阅读下面的内容（MATLAB 5.3）。

1. 系统目标文件

读者可以看到，在该页的代码生成面板（Code generation），有一个名为系统目标文件（System target file）的编辑框。System target file 用于指定代码的类型和生成代码的目标机器类型。单击编辑框右边的 browse 按钮可以浏览系统中可以使用的目标文件的完全列表。该列表还详细给出了各个目标文件的适用场合，例如，缺省的 grt.tlc 适用于普通代码目标。在选定了系统文件之后，用户就可以指定目标语言编译器（TLC），表 10-4 列出了常用的几个属性。

表 10-4 目标语言编译器的常用属性

选 项	说 明
-lpath	添加目录到搜索目标文件（后缀名.tlc）路径列表中去。
-m[N a]	当遇到错误时，报告的最大错误数目（缺省值是 5）。例如，-m3 说明了至多 3 个错误被报告，要报告所有的错误，请说明为-ma。
-d[glno]	说明调试模式（generate, normal 和 off）（generate, normal, or off），它的缺省值是 off。当-dg 被指定，一个.log 文件为用户的每一个 TLC 文件建立。当调试模式被激活，目标语言编译器显示目标文件的每一行遇到的次数。
-aVariable=expr	赋给在编译过程中被目标文件使用的变量一个指定的值（例如建立一个参数值对）。

这些属性的添加方式是在 System target file 域的文件名后直接添加。更多的选项设置可以在 Code Generation Options 对话框里获得，这个对话框是通过单击 Options 按钮打开的。

2. 内嵌参数

Inlining parameters 指模型中具有常数采样时间的模块在模型执行时被移走。这些模块的输出信号在模型一旦开始就建立。当这个检查框被选中，Simulink 自动地设置所有的内置参数为“不变常数”。一个参数一旦被设置为内嵌参数，就不能在仿真运行过程中动态的变化了，这样可以提高仿真的运行速度。Simulink4.0 中，内嵌参数的设置请看第六章的 Advance 页一节。

3. 保持 model.rtw 文件

当用户要修改目标文件时，就需要对照 model.rtw 文件。但在建立过程完成之后，model.rtw 文件一般会被删除，选中 Retain .rtw file 检查框可以避免这一点。

4. 模板 makefile

模板 makefile 在 Generate code only 检查框没有选中时使用。模板 makefile 唯一的说明了要生成的可执行文件的目标。用户可以在编辑框里输入自定义的 makefile 文件，但一般情况下，还是保持缺省值不变。

5. 建立命令

建立选项面板里的 make command 编辑框用来说明编辑建立命令。建立命令通常以 make \_rtw 开始（除非它被第三方建立命令所替代）。在用户按下 build 按钮之后，这个命令就被调用。用户还可以传递额外的参量到 make \_rtw。例如，当用户使用 grt\*.tmf 文件的一个时，

就可以改变优化选项，它的设置为：

```
make_rtw OPT_OPTS="compiler_specific_setting"
```

## 6. 系统目标文件浏览器

系统目标浏览器在用户按 **Browse** 按钮后出现，它给出了 RTW 支持的目标文件列表。因为 RTW 支持多种目标和代码格式，为了方便读者，RTW 提供了这个浏览器，让读者可以直接地选择适合自己应用的目标。

## 7. 选项按钮

在仿真参数对话框里有一个选项按钮。按下这个对话框会打开一个代码生成选项对话框，这个对话框会随用户在系统目标文件浏览器选择的系统目标文件变化而变化。图 10-11 显示了当系统目标文件选择为 `grt.tlc` 时对话框的样子。

对话框所支持的特性取决于所选择的目标，但一般都有下面典型的几种：

- Loop rolling threshold;
- Show eliminated statements;
- Verbose builds;
- Inline invariant signals;
- Local block outputs.

它们的具体作用解释如下：

(1) **Loop rolling threshold**。让用户设置使循环转换 (loop rolling) 发生时的门限 (一个正整数)。例如，设置这个门限值为 5，意味着如果你的算法被连续调用的次数超过 5 次，那么就把整个算法嵌到一个 `for` 循环中去。

(2) **Show eliminated statements**。选中这个检查框，将使得 RTW 把在优化时被删除的语句作为注释语句显示在生成代码里。

(3) **Verbose builds**。这个特性强迫命令行输出代码生成状况和编译器输出。

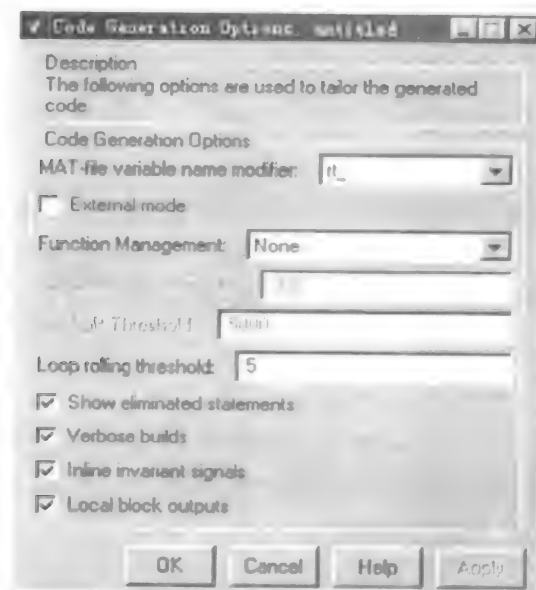


图 10-11 代码生成选项对话框

(4) Invariant signals。注意“invariant constants”和“invariant signals”并不是等同的。“invariant signals”指在 Simulink 仿真过程中保持不变的模块输出信号。例如在图 10-12 所示的模型中的 S3 信号就是一个不变信号。如果用户在 Code Generation Options 对话框里选择了 Inline invariant signals, Real-time Workshop 就把不变信号 S3 内嵌在生成的代码里。这和 Real-Time Workshop 页的 Inline parameters 不同,后者对应应用于不变常数,在这个例子中,如果用户选中了 Inline parameters 检查框, Real-Time Workshop 就内嵌两个常数和增益。

注意,对于内置不变信号,用户必须预先选定内置参数,如果 Inline parameters (内嵌参数) 检查框没有被选择, Inline invariant 选项就不会工作。然而,也有可能只内嵌参数,而不内嵌不变信号。

(5) Local block outputs。当这个检查框被选中时, Real-Time Workshop 试图放置尽可能多的模块输出信号作为局部变量。

(6) Online Help。把鼠标放在这些字段或检查框的任何一个时都将激活解释这个特性的用途的提示信息。

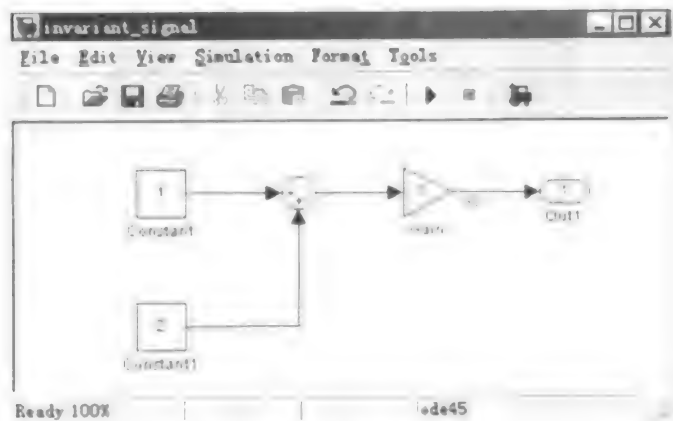


图 10-12 不变信号示例模型

在代码生成选项对话框里还可以进行的设置有: MAT-file variable name modifier 选择列表, 可以让用户决定是否要往生成的可执行程序运行后所生成的 MAT 文件内的变量名称添加前缀, 缺省值为“rt\_”, 它的另外一个值是“none”。External mode (外部模式) 检查框, 选中它使生成的实时代码支持外部模式。

通常还会有一个 Function manage 的多项选择列表。这个设置的作用是防止所生成代码的某个函数或者某个文件太长, 因为某些编译器对单个文件或者函数的长度有限制。如果在这里选择了 Function split 或者 File split, 那么 RTW 将会根据在下面的两个编辑框所设置的门限来决定是否将函数(文件)分割。

#### 8. Tunable parameters (可调参数)

当用户在 Advanced 页, 选中了 Inline parameters 检查框, Configure 按钮就变成了可用状态。单击它, 就可以打开模型参数配置对话框, 它如图 10-13 所示。

模型参数配置对话框支持下面的特性:

(1) Parameter tuning——去除 Variable 字段里的任何变量的内嵌特性, 可以通过单击 Add 按钮把变量名添加进去。也就是忽略掉所选变量的 Inline parameters 检查框, 使之成为

可调谐的，但其他的参数仍然保持内嵌特性不变。

(2) **Storage Class**——用户可以改变所选择变量的存储类型。下拉列表里提供的选项有：

- **Auto**——使 Real-Time Workshop 将变量保存在一个永久数据结构里，这是 RTW 的缺省存储类型选项：

- **Exported Global**——将变量声明为一个全局变量，这样就可以从生成代码的外部来访问它：

- **Imported Extern**——将变量声明为外部变量（extern），它必须从生成代码的外面进行声明：

- **Imported Extern Pointer**——将变量声明为外部指针（extern pointer），它必须从生成代码的外面进行声明。

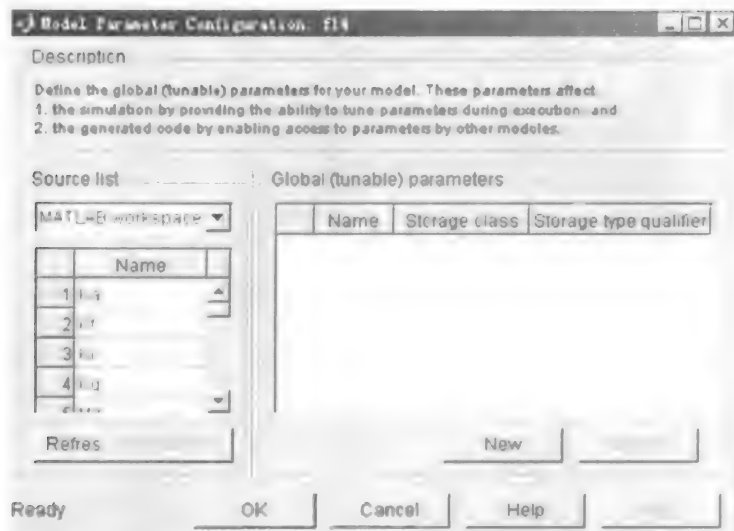


图 10-13 模型参数配置对话框

这些选项在读者想把 RTW 生成的代码和其他的 C 代码（不是 RTW 生成的）连接起来时，就非常有用。

(3) **Storage Type Qualifier**——这个编辑框输入时，如果对变量进行声明，要预先考虑任何字符串（例如，const）。注意 RTW 不会对字符串进行错误校验。

## 9. 信号属性

Real-Time Workshop 对信号支持和参数相同的存储类型选项。改变一个信号的存储类型，可以在模型里选中该信号后用 Edit 菜单下的 Signal Properties 命令来打开信号属性对话框。关于信号属性对话框，本书前面曾讲过一些。这里只介绍和 Real-Time Workshop 有关的选项，它们的位置在对话框的低端。图 10-14 显示了信号属性对话框。

在对话框里，可以设置的特性有：

(1) **Displayable**（可显示测试点）——选中这个检查框将使 Real-Time Workshop 为信号分配一个单独的全局的存储空间。这可以使测试时减少重写信号数据的可能性。注意一旦选择了这个选项会把 RTW storage class 强置为零。

(2) **RTW storage class**——可以改变所选中信号的存储类。右边下拉菜单可供选择的选项有：

- **Auto**——使 Real-Time Workshop 把信号存储为任何它适合的存储类。这是 RTW 的缺省存储类选项；
- **Exported Global**——把信号声明为可以从生成的代码外访问的全局变量；
- **Imported Extern**——把信号声明为外部变量（extern），于是它必须从生成代码外部来声明；
- **Imported Extern Pointer**——把信号声明为外部指针，于是它同样要在外部被声明。

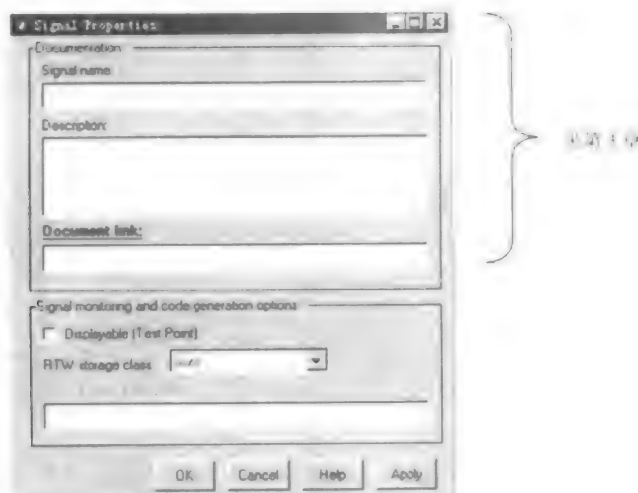


图 10-14 信号属性对话框

这些情况用在用户想把 RTW 代码和其他的 C 代码（即不是用 RTW 生成的代码）时，注意用户要负责把代码块正确地链接在一起。

(3) **C API for Signal Monitoring, RTW**——提供了开发独立于外部模式的信号监测的 C API See “C”。

## 10.4 外部模式

### 10.4.1 介绍

外部模式是 RTW 提供的一种支持在实时环境下动态改变参数的仿真模式。“External”是指这种模式涉及两种独立的计算环境、一个宿主和一个目标，外部模式允许这两个独立的环境相互通信。宿主是 MATLAB、Simulink 和 RTW 运行的计算机，而目标则是由 RTW 建立的可执行程序运行的计算机。使用外部模式，用户就可以改变模块参数并且在 Scope 模块查看和保存模块输出。外部模式实际上就是在 Simulink 和 RTW 生成的代码间提供一个通信通道。这个通信通道可以是网络协议，例如 TCP，或者是共享内存。

在外部模式中，宿主（Simulink）等待参数或者信号（来自外部）的变化，一旦接收到这种变化，它就发送请求目标接收参数变化的消息或者是上载信号数据，而目标响应这种请求之后，Simulink 再把新的参数值提供给目标机器。由此可见，外部模式通信是基于客户机/服务器体系结构的，其中 Simulink 是客户机，目标是服务器。

外部模式的主要功能有：

(1) 实时地修改或者调整模块参数。在外部模式，无论用户什么时候在模块图表里改变参数，Simulink 会自动的把它们下载到正在执行的目标程序中去。这样就允许用户不需重新编译就可以调整程序的参数。在外部模式，Simulink 变成了目标程序的图形前端。

(2) 使用多种模块和子系统，查看和记录模块输出。用户不需写特殊的接口代码，就可以监视或者记录从正在执行的目标程序输出的信号数据。外部模式允许用户定义数据从目标上载到宿主的条件。例如，数据上载可以由一个正向的过零点信号来触发，类似的，它也可以由用户手动触发。

外部模式的工作机制是在 Simulink 模型和 RTW 生成代码之间建立一个通信通道。这个通道由一个处理消息的物理传送的低层传输层来实现，而 Simulink 和生成代码都是和这个层独立的。这种设计使得不同的目标使用不同的传输层称为可能。例如，GRT 和 Tornado 目标使用 TCP/IP 来实现宿主和目标间的通信；而 xPC 目标支持 RS-232（串口）和 TCP/IP 通信；Real-Time Windows Target 则是通过共享内存来实现外部模式（Real-Time Windows Target 是和 RTW 独立的产品）。

这一节将向读者介绍：

- (1) 如何在自己的机器上建立外部模式，这时，宿主和目标在同一台机器上；
- (2) 如何使用包含在 Simulink 里的基于 TCP socket 的外部模式实现；
- (3) 参数调整；
- (4) 使用 Simulink Scope 模块进行信号查看和保存。

在学习本节之前，请读者先学习前面一节“代码生成和建立过程”。

#### 10.4.2 使用 grt（普通实时目标）的外部模式入门

这一小节将通过一个使用普通实时目标（GRT）的示例，一步一步的向读者介绍外部模式的入门知识。这个例子的宿主和目标处在相同的机器，它不需要读者准备另外的机器来运行可执行程序，您就可以体验一下什么是外部模式。如果读者的兴趣仅在此为止，那么这一小节以后的内容，就可以先放在一边。

在进入外部模式的建立之前，请读者先建立一个简单的模型来作为生成代码的模型。读者可以使用本书的示例，也可以自己建立一个模型，这不影响下面的介绍。图 10-15 显示了本书采用的简单模型。

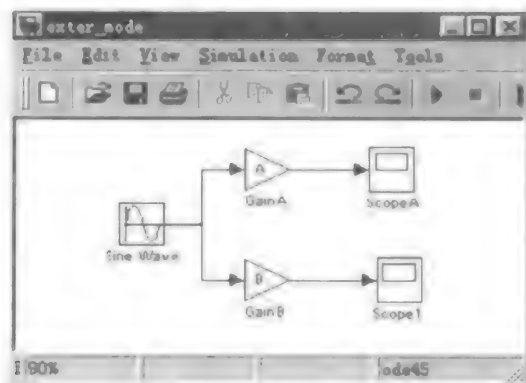


图 10-15 外部模式的示例模型

下面就来为这个模型建立外部模式目标程序，其步骤为：

(1) 使用 **simulation** 菜单下的 **External** 命令把仿真设为外部的。

(2) 打开仿真参数对话框。在 **Solver** 页，把 **Solver Options Type** 置为 **Fixed-step**（固定步长），并且把 **Fixed-step Size**（固定步长大小）置为 0.01，而仿真时间 **Stop Time** 依旧是缺省值保持不变，最后置 **Decimation** 为 1。

(3) 在 **Workspace I/O** 页，去掉 **Time** 和 **Output** 检查框的选中状态，因为在这个例子中，数据不需要保存到工作空间或者 MAT-文件中。

(4) 在 **Real-Time Workshop** 页，从 **Category** 菜单选择 **Target configuration** 选项。而至于系统目标文件（**System Target File**），缺省情况下，应该指定为普通实时目标（即 **grt.tlc**），所以就不用修改。如果没有设定为 **GRT**，读者可以从 **Browse** 打开的系统目标文件列表里选择它。最后完成的 **Real-Time Workshop** 页应该如图 10-16

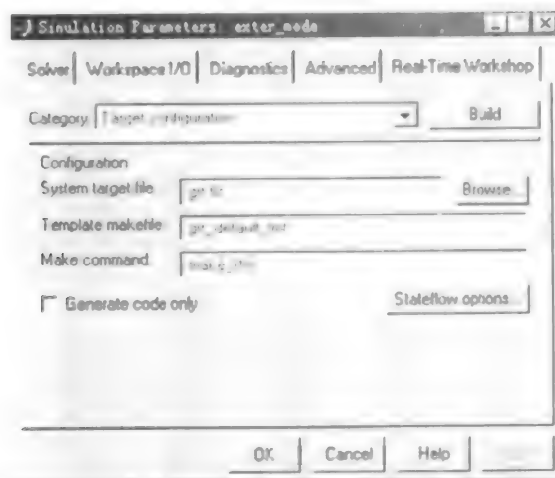


图 10-16 最终的 RTW 页

(5) 从 **Category** 菜单选择 **GRT code generation options** 选项，并在相应出现的选项面板里选中 **External mode** 检查框（图 10-17），这样 RTW 就会生成支持外部模式的代码。

(6) 选择 **Advanced** 页，确保 **Inline parameters** 检查框未被选中。外部模式不支持内嵌参数选项。

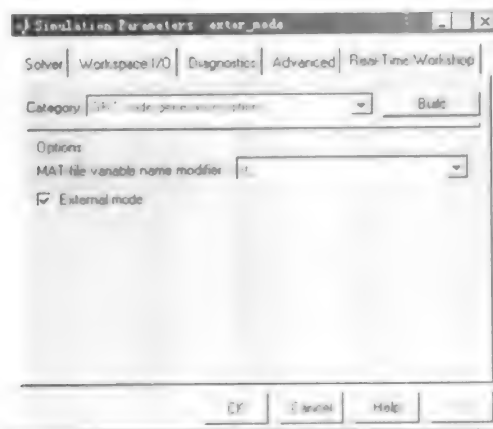


图 10-17 设置 GRT 代码生成选项



(7) 设置外部模式的特性。用 Tools 菜单下的 External Mode ControlPanel 命令打开模式控制面板，读者可以通过它方便地浏览外部模式要用到的所有特性。外部模式面板能让用户配置宿主和目标之间的通信机制，信号监视以及数据记录等特性，也能让用户将 Simulink 模型连接到目标程序，开始和结束模型代码的执行。图 10-18 是外部模式控制面板的样子。

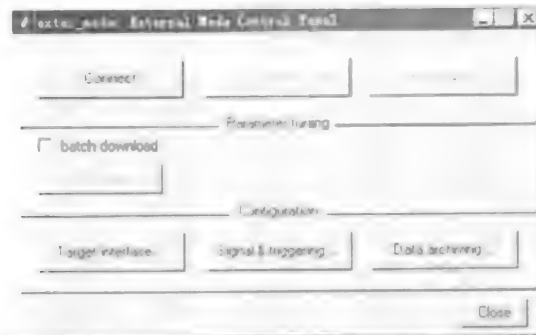


图 10-18 外部模式控制面板

图中的上面 4 个按钮在读者运行了实时程序后使用，而下面的 3 个按钮将分别打开 3 个不同的对话框。它们分别是：

- Target Interface 按钮打开 External Target Interface 对话框（外部接口对话框），它用来配置外部模式的通信通道。
- Signal & triggering 按钮打开 External Signal & Triggering 对话框（外部信号和触发对话框），它配置哪个信号被查看并且它们如何被触发。用户在开始执行实时程序前必须先在对话框里选择信号。触发一词表示信号何时被捕获和显示。
- Data archiving 按钮打开 External Data Archiving（外部数据记录）对话框。

(8) 按 Target Interface 按钮，打开目标接口对话框，图 10-19 是对话框的外观。Simulink 要求用户在外部模式下运行模型前，必须先配置目标接口和设置各种信号触发和数据记录选项。缺省情况下，作为外部接口的 MEX-文件应该设置为 ext\_comm。这个文件支持使用 TCP 通信协议在宿主和目标间通信。MEX-文件的参量可以留作空白。设置完毕后，请按 ok 按钮，退出外部目标接口对话框。

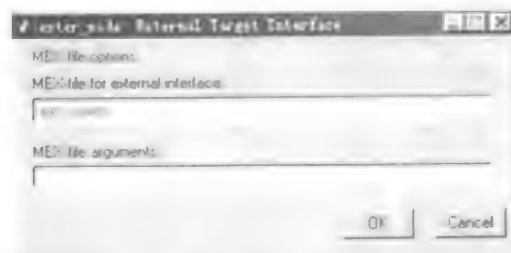


图 10-19 目标接口对话框

(9) 设置信号触发和数据记录。这是启动外部模式的先决条件之一，读者可以单击 Signal & triggering 按钮来打开外部数据保存配置对话框。本例中，读者可以按下面来配置这个对话框：

- 单击选择 Select All，这会选中示例模型中的两个模块，使我们能够实时的查看输出的信号。



- 在触发面板, 设置 Trigger Source 为 manual (手动), Trigger Mode 为 normal, 持续时间为 1000, 以及延迟为 0 (没有延迟)。

- 选中 Arm when connect to target 检查框。

(10) 关闭外部模式控制面板, 并保存模型。

(11) 单击 RTW 页上的 Build 按钮在外部模式下生成代码, 并建立可执行程序。在按 Build 按钮之前, 用户必须先在 MATLAB 工作空间设置增益 A 和 B 的值, 这里不妨使用 A=3, B=3 (关于模块参数估值请看第四章)。至此, 建立外部模式程序的整个过程就完成了。

☞ 注意, 在这个例子中, 数据记录选项 (data archiving) 不需要设置。在用 build 命令生成代码前, 请读者先检查是否给 A、B 赋值, 以及是否设置了环境变量。而上面的第 9 步操作对可执行程序的建立没有影响, 它可以放在建立好目标可执行程序后进行。

接下来就可以运行建立的目标可执行程序了。请读者在 MS-DOS (UNIX 对应的是 Xterm) 命令行下输入

```
modelname -tf inf -w
```

或者将前面的命令加上 “!”, 后面加一个 “&”, 就可以直接在 MATLAB 命令窗口输入。例如

```
!modelname -tf inf -w &
```

在这里, 模型名称是 `exter_mode`。命令中后面的 -tf 开关将重载 Simulink 模型仿真参数对话框中设置的结束时间。把结束时间值设为 tftf, 表示模型的运行时间是不确定的, 模型会一直运行至接收到一个来自 Simulink 的消息为止。-w 开关的意思是让目标程序在接到宿主发出的开始实时代码的消息之前, 处于等待状态。如果用户想从程序开始时就观察它的输出, 或者是在程序执行前改变参数, 这个开关就是必须的。

然后, 选择 Simulation 菜单下的 Connect to target 命令, 这个命令初始化 Simulink 和目标代码之间的握手。当 Simulink 和目标代码连接上之后, 菜单下的 Start real-time code 命令就会变成可用状态, 而 Connect 按钮的标题会变为 “Disconnect”。选择这个 Start real-time code 命令之后, 就可以在 Simulink 模型的 scope 模块查看仿真的结果。运行的结果为。

读者可以动态地改变增益模块的增益值, 或者是在 MATLAB 工作空间设置新的增益值。在后一种情形, 请键入 Ctrl-D 命令或者用 Edit 菜单下的 Update Diagram 命令, 在改变 MATLAB 变量值后来更新模型, 按这种改变参数的方法称为 downloading。

类似的, 读者可以改变正弦波的频率、幅度或者相位。但是读者不能改变正弦波模块的采样时间。因为模块采样时间是模型结构定义的一部分, 因而也是生成代码的一部分。所以, 如果读者要改变一个模块的采样时间, 那就必须重新编译模型, 重新建立可执行程序。

### 10.4.3 外部模式 GUI

外部模式 GUI 是 RTW 提供的一个扩展图形用户界面, 它是个独立的窗口, 用来支持外部模式里的特性。这 4 个窗口分别是:

(1) 外部模式控制面板;

- (2) 外部目标接口对话框;
- (3) 外部信号和触发对话框;
- (4) 外部数据记录对话框。

下面将逐一介绍这些特性, 尽管有一些我们在前一节已经提到过, 但这一节将会给出更详细的内容。

### 1. 目标接口

按 Target Interface 按钮将激活 External Target Interface 对话框。它用来设置 MEX 文件选项:

(1) 作为外部接口的 MEX 文件——外部接口文件的名称。缺省值是 `ext_comm`, 它是为使用 GRT 和 Tornado 目标提供的基于 TCP/IP 的外部接口。而 Real-Time Windows Target 使用的接口文件是 `win_tgt`, 对于客户或者第三方目标, 它们可能使用其他的接口 MEX 文件。

(2) MEX 文件参量——外部接口文件的参量。例如, 基于 TCP 的外部接口 MEX 文件 `ext_comm`, 包含 3 个附带参量: 目标的网络名、冗长等级和 TCP 服务器端口数目。缺省情况下, 这个域留作空白。对于 `ext_comm`, 读者可以在编辑框内输入本地目标名称, 长度等级和端口号 17725, 即: `'localtargetname', 1, 17725`

### 2. 外部信号和触发

图 10-20 是外部信号和触发窗口的示意图, 这个窗口提供以下的支持:

- (1) 信号选择——说明哪个 scope 模块在外部模式是激活状态。
- (2) 触发选择——设置这个信号何时在 Scope 模块显示数据。
- (3) 触发选项——配置如何并且何时在 Scope 模块显示数据。



图 10-20 外部信号和触发窗口

外部模式的信号选择只限于 Scope 模块的输入信号, 用户模型中的任何 Scope 模块都会出现在信号选择列表里。Select all 和 Clear all 按钮分别选择和选择不选择所有列出的信号。一个信号被选中时, 那在模块的名称左边就会出现一个 X。当一个信号被选择时, 如果用鼠标单击信号列表中的模块名称, 静态按钮将会被激活。

读者可以通过在信号列表中选择—一个信号，然后按下 **Trigger signal** 按钮，来选中触发信号。如果一个信号被选中为触发，那么字母“T”会出现在模块名字的左边。注意，这个标记只在触发源被置为信号触发选项时，才会出现。在对话框的下端，有不同的选项可以使用：

(1) **Source**——触发源，有两种选择：**manual** 或者 **signal**。选择 **manual** 使得外部模式，在外部模式控制面板的 **Arm trigger** 按钮一被按下，就开始记录数据。选择 **signal** 则告诉外部模式在选中的触发信号满足触发条件（按指定的方向滑过触发门限）时就开始记录数据。

(2) **Duration**——持续时间。这个时间也就决定了外部模式在设定的触发事件发生后，记录数据持续的基准速率时间步的数目。例如，模型中最快的速率是 1 秒钟，对于频率为 1HZ 的信号，如果记录的 **Duration** 设为 10 秒，于是外部模式将会收集 10 个采样。如果 2HZ 的信号被记录，在同样的持续时间下，那么 5 个采样被收集。

(3) **Mode**——模式。也有两种选择：**normal** 或者 **one-shot**。在 **normal** 模式，外部模式在每个触发事件发生后自动重新准备触发器。而在 **one-shot** 模式，外部模式在用户每准备好触发器一次，就只收集一个缓存的数据。

(4) **Delay**——延迟时间。用户可以在触发器设置一个延迟，其数值是基本速率的仿真步的数目，可以为正也可以为负，表示数据收集相对于触发发生的延迟时间。一个负的时延就对应着前面的一个触发。

(5) **Arm when connect to target** ——当连接到目标时准备好触发器。这个检查框告诉外部模式在用户连接到目标时，就自动准备好触发器。一旦触发器被准备好，外部模式就监视触发信号是否具有指定的触发条件。

在触发信号选择好之后，触发信号面板将变为可用状态，用户可以在它上面定义触发条件，以及设置信号所处的 **Port**（端口）和 **Element**（元素），因为前面选择触发信号，是根据所属模块来标记的。缺省情况下，所设定的触发模块的第一个端口的任何元素都能够导致触发器被触发。用户可以调整 **Port** 值和触发面板右端的 **Element** 域来改变这些行为。**Port** 域可以是一个数字或者关键字 **last**，而 **Element** 域则能接收一个数字或者关键字 **any** 和 **last**。

触发信号面板还可以设置触发信号触发的条件，供选择的选项有：

(1) **Direction**——触发方向。它包括：**rising**（上升）、**falling**（下降）或者 **any**（任何）。触发信号只有在按设定的方向滑过门限值才会产生激活触发器，也可以理解为触发事件的类型，它们和触发子系统里的定义是一致的。

(2) **Level**——说明触发信号的门限值。触发信号只有在按 **Direction** 所设定的方向滑过这个值时才会激活触发器。缺省情况下 **level** 的值设为 0，如果它被设置为 1，那么，将使触发器在触发信号按设定的方向滑过 1 时才触发。

(3) **Hold-off** ——只在 **normal** 模式下有用。其单位是基准的时间步，它表示一次触发事件和触发器准备好的时间间隔。

### 3. 数据记录

这个窗口支持外部模式的以下特性：

(1) **Directory notes**——目录层次注释。设置它，可以通过 **Edit directory note** 按钮打开 MATLAB 编辑器（M 文件），要保存在特定目录的注释就可以放在编辑窗口里。

(2) **File notes**——文件层次注释。设置它，可以通过按 **Edit file note** 按钮打开文件查找窗口。缺省下选中的文件是最近写过的文件。选择任何的 **MAT** 文件打开一个编辑窗口，然

后就可以添加或者编辑注释，并保存为自己的文件。

(3) Data archiving——自动的数据记录和存储。按 **Enable Archiving** 按钮就激活了外部模式的自动数据档案特性。为了了解数据档案特性工作的原理，读者有必要知道在档案特性被禁止时数据获取的工作机制。这有两种情况：**one-shot** 和 **normal** 模式。在 **one-shot** 模式，在一个触发事件发生后，每一个被选择为触发器的 **Scope** 模块只会在仿真快要结束时，才写数据到工作空间。如果仿真过程中，还有另外一个 **one-shot** 事件被触发，那么工作空间中已存在的数据将会被擦除重写。在 **normal** 模式，外部模式在每一个触发事件发生后，就自动地重新准备触发器。所以，读者可以把 **normal** 模式看成一串的 **one-shot**，其中的每一个 **one-shot**，除了最后一个之外，都会引起一个即时结果。因为触发器可以在任何时候触发，写即时结果到工作空间会导致对工作空间变量不可预测的重写。所以，缺省的行为是在最后一个 **one-shot** 触发后才把结果写入工作空间，而即时结果被摒弃。如果用户知道在触发器里有足够的时间间隔来监视即时结果，那么用户就可以选中 **Write to intermediate workspace** 检查框来重定义这种行为。注意这个选项不保护工作空间数据不被后面的触发器重写。

**External Data Archiving** 对话框的选项支持保存数据自动写入（包括即时结果）到磁盘。数据档案提供下面的设置：

- **Directory**——设定保存数据的目录，如果选择了 **Increment directory when trigger armed** 检查框，外部模式会自动添加一个后缀到目录名上。

- **File**——指定数据保存的文件名，在 **Increment file after one-shot** 被选中后，外部模式也会自动添加一个后缀到文件名后。

- **Increment directory when trigger armed**——外部模式在每次 **Arm trigger** 按钮被按下时，都会使用一个不同的目录来存放数据保存文件。目录的名称是通过在设定的目录后添加一个按升序增加的数字来命名（这种文件称为增序文件）的，例如，**dirname1**，**dirname2** 等等。

- **Increment file after one-shot**——新的数据缓存被保存到增序文件中，如 **filename1**，**filename2** 等等，记住这在正常模式下是自动发生的。

- **Append file suffix to variable names**——无论何时外部模式增加文件名的序号，每一个文件保存的变量都会被加上所处文件的序号作为后缀。例如文件 **file\_1**，**file\_2** 文件里的数据变量 **xdata** 分别为 **xdata\_1**，**xdata\_2**，于是每个文件内的变量名在整个工作空间里就是单一的。这主要是为了便于在 **MATLAB** 比较变量。如果不是单一的，那么在分析时，新文件里的变量就会在 **MATLAB** 工作空间，覆盖前面文件里的变量。图 10-21 显示了在 **archive** 被使能下 **External Data Archiving** 对话框的样子。

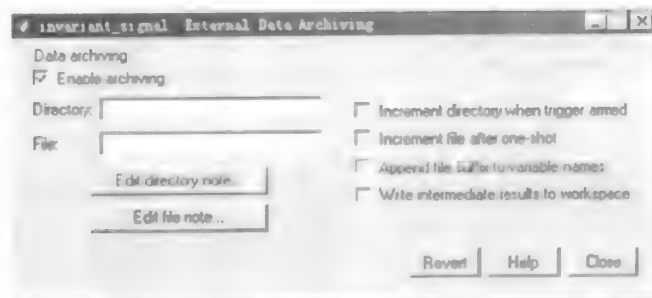


图 10-21 External Data Archiving 对话框

#### 10.4.4 外部模式的 TCP/IP 实现

这一节来介绍如何使用 RTW 实现基于 TCP/IP 的客户机/服务器的计算模式。在读者的目标系统支持 TCP/IP 的前提下，读者就可以使用由 RTW 生成代码提供的基于 Socked 的外部模式实现。

低层的传输层负责消息的物理传输，Simulink 和模型代码和这一层都是独立的，它们被分隔在分离的代码块中，例如，格式，发送，接收消息以及数据包。

读者知道，使用 Simulink 外部模式，必须进行下面的操作：

- (1) 在外部目标接口对话框，设定外部接口 MEX-文件的名称。缺省情况下，这个文件是 ext\_comm。
- (2) 配置模板 makefile，以便它能为 TCP/IP 服务器代码链接到更合适的源代码，并且定义建立生成代码所必须的编译器标识。
- (3) 建立外部程序。
- (4) 运行外部程序。
- (5) 将 Simulink 设为外部模式并且连接到目标。

介绍外部模式的 TCP/IP 实现，也将遵循这样的步骤。图 10-22 显示了基于 TCP/IP 的外部模式实现的结构。

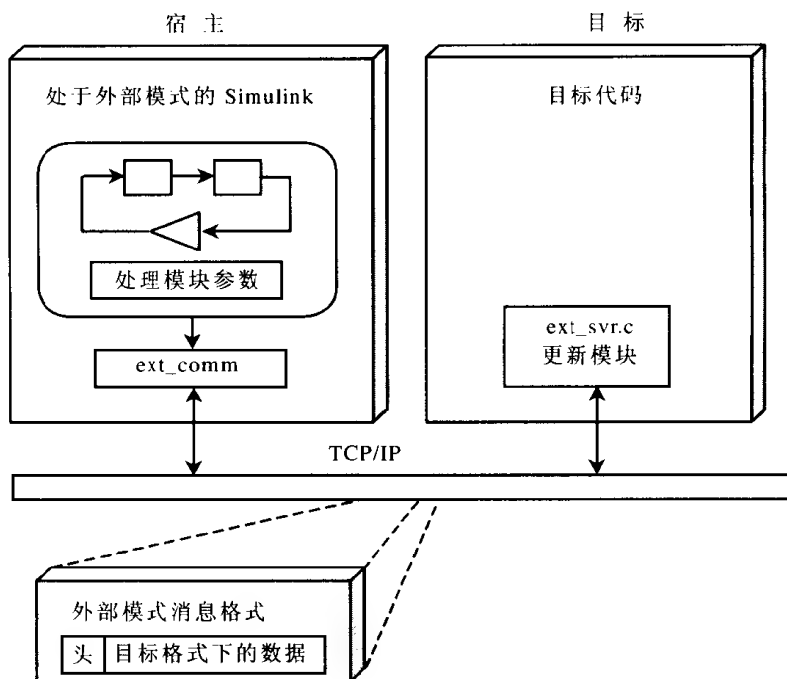


图 10-22 基于 TCP/IP 的外部模式实现的结构

建立外部程序的第一件事是设置外部接口 MEX-文件，它的缺省设置是 ext\_comm，它实现了基于 TCP/IP 的通信。ext\_comm 具有三个额外的参量，读者可以在外部接口文件对话框的参数对话框设置这些参量。它们如下所述。

- (1) 目标网络名：运行外部程序的计算机的网络名称。缺省情况下，它的取值为运行

Simulink 的计算机的名称, 这就是宿主和目标相同的外部模式。这个名称可以是一个用单引号限定的字符串, 例如 'myPuter', 也可以是单引号限定的 IP 地址, 例如 '148.27.151.12'。

(2) 详细等级: 控制数据传送过程中所显示信息的详细等级。取值为 0 或者 1, 它们的意义如下:

0——无信息;

1——详细的信息。

(3) TCP/IP 服务器端口数: 缺省值是 17725。读者在必要时, 可以把它设为一个处于 256 和 65535 之间的值, 来避免端口冲突。

读者在设定上面的选项时, 必须按照次序进行。例如, 如果读者想设置信息的详细等级 (第二个参量), 那么你必须同时要同时设定目标机器名 (第一个参量)。

接下去的操作和前面讲的差不多。为了能使外部模式代码生成, 读者要在 Real-Time Workshop 页的 target-specific code generation options 面板选中外外部代码检查框。

外部模式代码生成之后, 就要运行外部模式代码, 读者可以使用前面的两种方式: 在 MS-DOS 命令行输入 `modelname -tf inf -w`, 或者直接在 MATLAB 命令窗口输入! `modelname -tf inf -w &`。关于命令的参量说明请见 10.3.2。

如果 Simulink 模块图表和外部程序不匹配, Simulink 显示一个错误对话框告诉用户检查不匹配, 可能在生成代码后, 模型已经被修改过。这种情况的解决方式, 是重新根据新的模块图表建立外部程序。如果外部程序不运行, Simulink 也会给出一个错误提示, 表明无法连接到外部程序。

## 10.5 RTW 代码库

为了方便用户构建实时系统模型和生成代码, RTW 提供了一个函数库, 读者可以用 SIMULINK 库浏览器看到这个库。图 10-23 是它的示意图。

从图中读者可以看到, RTW 库有 5 个子库组成, 它们分别是:

(1) Custom Code Library —— 模块集合, 它允许用户将自定义的代码插入到生成的源代码和用户模型相关的函数;

(2) DOS Device Drivers——为 DOS 目标设计的模块;

(3) InterruptTemplates——用户可以把它作为模板生成自己的异步中断的模块集合;

(4) S-Function Target——这个模块用于和 Real Time W S-函数代码格式结合使用;

(5) VxWorks Support——支持 VxWorks (Tornado) 的模块集合。

### 10.5.1 Custom Code Library (自定义代码库)

自定义代码库的作用是为用户提供一些可以在 RTW 生成的代码里放置自己代码 (可以用 C 也可以用 Ada 编写) 的模块。对于 C 和 Ada 自定义代码库, 都有两个自定义代码子库:

(1) Custom Model Code (自定义模型代码);

(2) Custom Subsystem Code (自定义子系统代码)。

这两个子库包含的模块, 用户都可以在里面放置自己的代码, 这些模块分别指向特定的

模型文件和子系统。

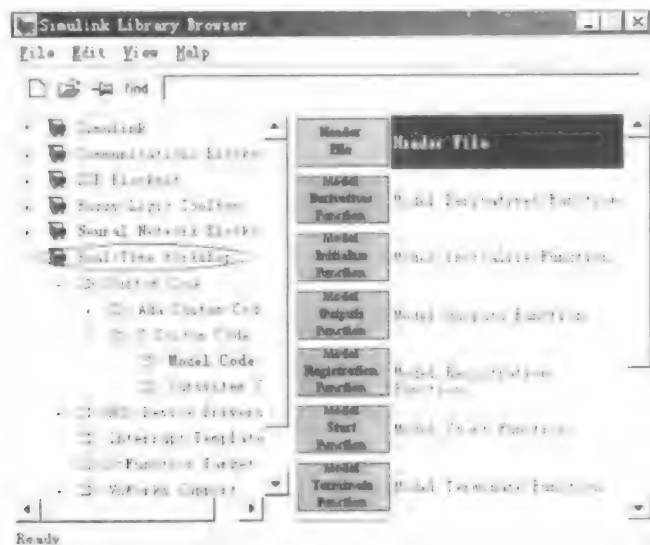


图 10-23 RTW 库

### 1. 自定义模型代码

自定义模型代码包含了 10 个模块，它们的编辑域使用户可以在生成的模型文件和函数中插入自定义代码。读者可以打开这个子库来观看这些模块，在字库中用右键上的命令打开的窗口如图 10-24 所示。其中，在第一行的 4 个模块的作用分别是分别在下面的 4 个文件的顶部和底部插入自定义代码。描述如下：

- (1) model.h ——Header File 模块；
- (2) model.prm——Parameter File 模块；
- (3) model.c——SourceFile 模块；
- (4) model.reg——Registration File 模块。

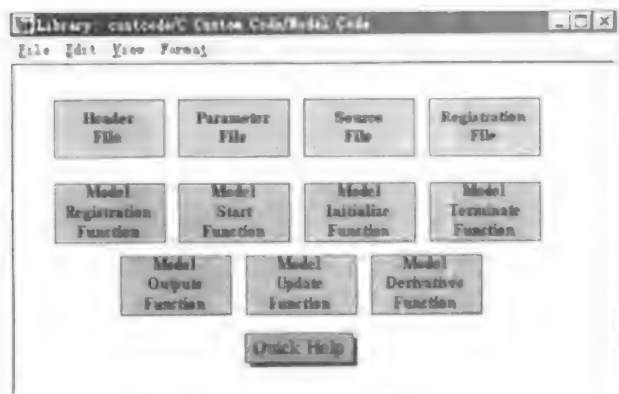


图 10-24 自定义模型代码

而第二行和第三行的 6 个模块，用户可以在它们的编辑域编辑要插入在下面所述的模型函数中的代码：

- (1) Registration function—Registration Function 模块；
- (2) MdlStart—MdlStart Function 模块；

- (3) MdlTerminate—MdlTerminate Function 模块;
- (4) MdlOutputs—MdlOutputs Function 模块;
- (5) MdlUpdate—MdlUpdate Function 模块;
- (6) MdlDerivatives—MdlDerivatives Function 模块。

## 2. Custom Subsystem Code (自定义子系统代码)

Custom Subsystem Code 库提供了 8 个模块, 用户可以使用它们把代码插入到系统函数中, 图 10-25 是这个子库的样子。



图 10-25 Custom Subsystem Code 子库

这 8 个模块分别是:

- (1) Subsystem Start 模块;
- (2) Subsystem Initialize 模块;
- (3) Subsystem Terminate 模块;
- (4) Subsystem Enable 模块;
- (5) Subsystem Disable 模块;
- (6) Subsystem Outputs 模块;
- (7) Subsystem Update 模块;
- (8) Subsystem Derivatives 模块。

这些子系统模块所在的位置决定了里面的自定义代码在模型代码中的位置。换言之, 代码对所选择的子系统是本地的。例如, Subsystem Outputs 模块在模块处于模型的顶层时, 就把代码放置在 mdlOutputs 函数中, 但如果它处于一个使能子系统内, 那么代码就被放置在这个系统的输出函数中。

对于一个触发子系统, 其中各输出函数的次序为:

- (1) Output entry;
- (2) Output exit;
- (3) Update entry;
- (4) Update exit code。

### 10.5.2 使用自定义代码模块示例

下面将举一个例子来演示如何使用自定义代码模块。这个例子将通过 MdlStart Function 模块将代码插入到 MdlStart 函数中, 图 10-26 是所用模型的图表, 其中插入了一个 MdlStart



Function 模块。

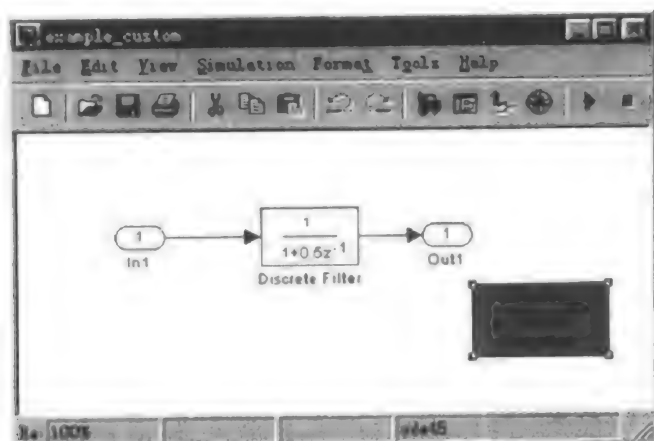


图 10-26 自定义代码模块使用示例模型

双击 MdlStart Function 模块，将打开图 10-27 所示的对话框，请按对话框所示，在相应的编辑域里输入代码。当然，读者可以任意地编译域输入代码。

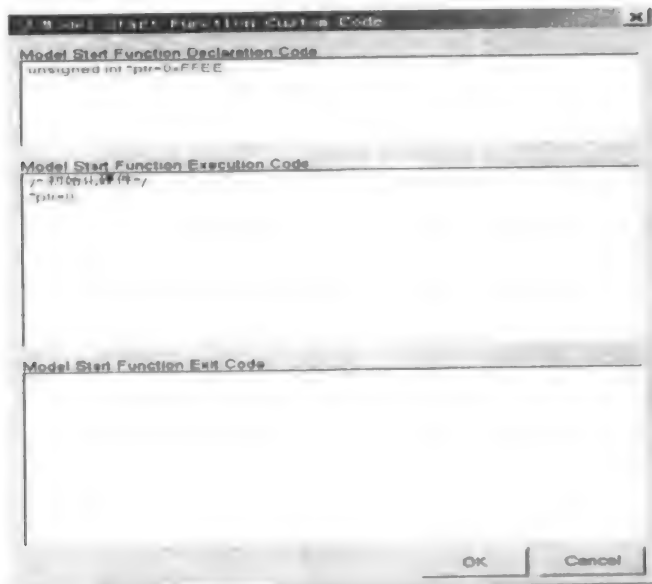


图 10-27 在 MdlStart Function 对话框输入代码

在声明代码编辑框里输入的代码为：“unsigned int \*ptr=0xFFEE;”，在执行代码编辑框里输入的两行代码为：“/\*初始化硬件\*/ ”和“\*ptr=0”。

于是，根据本例生成的代码的 MdlStart 函数就包含下面的代码：

```
void MdlStart (void)
{
/* user code (Start function Header) */
/* System: <Root> */
unsigned int *ptr = 0xFFEE;
```

```
/* user code (Start function Body) */  
/* System: <Root> */  
/* 初始化硬件 */  
*ptr = 0;  
/* state initialization */  
/* DiscreteFilter Block: <Root>/Discrete Filter */  
rtX.d.Discrete_Filter = 0.0;  
}
```

从中可见，在 MdlStart 函数模块对话框里输入的代码被直接嵌入到生成的代码中去了。

RTW 还提供了一个中断模板库，支持用户对同步或者异步事件的处理，包括中断服务方法 (ISRs)。使用库里的模块，读者就可以建立能够处理异步事件的模型，这些事件包括硬件产生的中断，和异步读写操作等等。这方面的信息，已经超出本书的范围，有兴趣的读者可以参考 RTW 的帮助文档。



## 附录 MATLAB 函数参考

本节按函数功能的分类，列出了 MATLAB6.0 的函数的简单说明，至于函数的具体用法，请读者自行查阅 MATLAB 的帮助。整个小节的内容安排为，在表 A-1 给出各功能子类的索引，而剩余的表格则分别对应着一类函数的简要说明。

表 A-1 各类函数的索引

函数目的	索引	函数目的	索引
通用目的命令	A-2	操作符和特殊字符	A-3
逻辑函数	A-4	语言结构和调试	A-5
特殊矩阵	A-6	基本数学函数	A-7
基本矩阵和矩阵处理	A-8	坐标系统转换	A-9
特殊数学函数	A-10	矩阵函数和数值线性代数	A-11
数值分析和傅立叶变换函数	A-12	多项式和插值函数	A-13
范函——非数值方法	A-14	稀疏矩阵函数	A-15
音频处理函数	A-16	字符、字符串函数	A-17
文件输入/输出函数	A-18	位逻辑运算函数	A-19
单元数组函数	A-20	多维数组函数	A-21
绘图和数据可视化	A-22		

表 A-2 通用目的函数

管理命令和函数			
addpath	将目录加到 MATLAB 的搜索路径	pathtool	打开查看和修改 MATLAB 路径的 GUI
doc	在 help 浏览器显示帮助文档	profile	打开 M 文件简化器
genpath	生成一个路径字符串	type	列出 M 文件
help	在命令窗口显示 MATLAB 帮助文档	rehash	刷新函数和文件系统 cache
what	M、MAT、MEX 文件的目录列表	rmpath	从 MATLAB 搜索路径中去掉目录
helpdesk	显示帮助浏览器	support	打开 MATLAB 文档支持主页
lasterr	上一个错误信息	lookfor	在所有的 help 条目搜索特定的关键字
lastwarn	上一个警告信息	path	显示 MATLAB 的目录搜索路径
管理变量和工作空间			
clear	从内存中清除变量和函数	disp	
who	列出当前变量	whos	为列出当前变量详细信息

续表

管理变量和工作空间			
length	向量的长度	retrieve	从硬盘获得变量
memory	存储空间限制的帮助	mlock	阻止 M-文件清除
munlock	允许 M-文件清除	openvar	在数组编辑器里打开工作空间变量
pack	巩固工作空间内存	save	保存工作空间变量在硬盘里
saveas	使用特定的格式保存图或模型	size	数组的尺度
who,whos	列出工作空间的变量	workspace	显示工作空间浏览器
控制命令窗口			
cedit	设置命令行编辑	clc	清除命令窗口
home	光标置左上角	format	设置输出格式
echo	在执行时显示 M 文件	more	在命令行控制分页输出
与文件和操作系统有关的命令			
beep	产生一个蜂鸣声音	cd	改变工作目录
checkin	阻止文件进入源控制系统	checkout	阻止文件出源控制系统
cmopts	获得源控制系统名称	copyfile	拷贝文件
customverctrl	允许自定义源控制系统	delete	删除文件和图形对象
diary	把任务保存到硬件文件	dir	显示目录列表
dos	执行一个 dos 命令并返回结果	edit	编辑一个 M-文件
fileparts	获得文件名部分	info	显示联系信息, 工具箱 Readme 文件
fullfile	从部分建立全文件名	matlabroot	获得 MATLAB 安装的根目录
inmem	在内存中的函数	open	根据扩展名打开文件
mkdir	建立新目录	pwd	显示当前目录
tempdir	返回系统的临时目录的名称	tempname	临时文件的单一名称
undocheckout	取消源控制系统的上次校验	!	执行操作系统命令
启动和退出 MATLAB			
finish	MATLAB 结束 M-文件	exit	终止 MATLAB
matlab	启动 MATLAB (仅对 UNIX)	matlabrc	MATLAB 启动 M-文件
quit	终止 MATLAB	startup	MATLAB 启动 M-文件

表 A-3

操作符和特殊字符

操作符和特殊字符			
+	加	-	减
*	矩阵乘法	.*	数组乘法 (按元素进行, 以下同)
^	矩阵幂	.^	数组幂
\	左除或反斜杆	/	右除或斜杆
/	数组除	kron	Kronecker 张量积

续表

操作符和特殊字符			
:	冒号	()	圆括号
[]	方括号	.	小数点
..	父目录	...	继续
,	逗号（分割多条命令）	;	分号（禁止结果的显示）
%	注释	!	感叹号
'	转置或引用	=	赋值
==	相等	<>	关系运算符
&	逻辑与		逻辑或
~	逻辑非	xor	逻辑异或

表 A-4

逻辑函数

逻辑函数			
all	测试所有的元素是否都是非零值	any	测试是否存在非零值
exist	检测一个变量或者文件存在	find	找出非零元素的值和下标
is*	检测状态	isa	检测一个给定类的对象
iskeyword	检测字符串是否为 MATLAB 关键字	isvarname	检测一个字符串是否为有效的变量名
logical	将数值转换为逻辑值	mislocked	如果 M-文件不能被清除返回真值

表 A-5

语言结构和调试

MATLAB 作为编程语言			
builtin	以重载方法执行内置函数	evalc	解释大写的 MATLAB 表达式
eval	解释包含 MATLAB 表达式的字符串	evalin	估算工作空间的表达式
feval	函数估值	function	函数 M-文件
global	定义全局变量	nargchk	检查输入参量的数目
persistent	定义永久变量	script	脚本 M-文件
控制流			
break	终止 for 循环和 while 循环的执行	case	条件切换
catch	开始 catch 块	continue	将控制传给 for 或者 while 循环的下一次运算
else	条件执行语句	elseif	条件执行语句
end	结束 for, while, switch 等复合语句	error	显示错误消息
for	重复执行语句设定的次数	if	条件执行语句
otherwise	switch 语句的缺省部分	return	从被调用函数返回
switch	根据表达式在几种情况切换	try	开始 try 语句块
warning	显示警告信息	while	在某种条件满足时重复执行语句

续表			
交互输入			
input	请求用户输入	keyboard	在 M-文件调用键盘
menu	为用户的输入建立一个选择菜单	pause	暂时终止执行
面向对象编程			
class	建立对象或者返回对象的类	double	转换为 double 精度
inferiorto	下层对象关系	inline	构造内嵌对象
int,int16	转化为有符号整数	isa	检测一个给定类的对象
loadobj	为用户对象扩展载入 (load) 函数	saveobj	用于对象的保存过滤器
single	转换为单精度	superiorto	超级类关系
uint8,uint16	转化为无符号数	uint32	转化为无符号数
调试命令			
dbclear	清除断点	dbcont	重新开始执行
dbdown	改变局部工作空间上下文	dbmex	启用 MEX-文件调试
dbquit	退出调试模式	dbstack	显示函数调用堆栈
dbstatus	列出所有的断点	dbstep	从一个断点执行一行或者更多行
dbstop	在 M-文件函数设置断点	dbtype	列出 M-文件的行数
dbup	改变局部工作空间上下文		
函数句柄			
function_handle	MATLAB 数据类型, 是函数的句柄	functions	返回关于函数句柄的信息
func2str	从一个函数句柄构建一个函数名	str2func	从一个函数名字符串构建函数句柄

表 A-6 特殊矩阵

特殊矩阵			
compan	友矩阵	gallery	几个小的矩阵
hadamard	Hadamard 矩阵	hankel	Hankel 矩阵
hilb	Hilbert 矩阵	invhilb	逆 Hilbert 矩阵
magic	魔方矩阵	pascal	Pascal 矩阵
toeplitz	Toeplitz 矩阵	wilkinson	Wilkinson 特征值测试矩阵

表 A-7 基本数学函数

基本数学函数			
abs	绝对值和复数的模长	acos,acosh	反余弦和反双曲线余弦
acot,acoth	反余切和反双曲余切	acsc,acsch	反余割, 反双曲余割
angle	相角	asec,asech	反正割, 反双曲正割
secant	正切	asin, asinh	反正弦, 反双曲正弦

续表

基本数学函数			
atan, atanh	反正切和反双曲正切	tangent	正切
atan2	四象限反正切	ceil	向着无穷大舍入
complex	从实部和虚部建立一个复数值	conj	复数配对
cos, cosh	余弦和双曲余弦	csc, csch	余切和双曲余切
cot, coth	余切和双曲余切	exp	指数
fix	朝零方向取整	floor	朝着负无穷取整
gcd	最大公因数	imag	复数值的虚部
lcm	最小公倍数	log	自然对数
log2	以 2 为底的对数	log10	常用对数
mod	有符号的求余	nchoosek	二项式系数和全部的组合数
real	复数的实部	rem	相除后求余
round	取整为最近的整数	sec, sech	正切, 双曲正切
sign	符号函数	sin, sinh	正弦和双曲线正弦
sqrt	平方根	tan, tanh	正切和双曲正切

表 A-8

基本矩阵和矩阵操作

基本矩阵和数组			
blkdiag	从输入参量建立块对角矩阵	eye	单位矩阵
linspace	产生线性间隔的向量	logspace	产生对数间隔的向量
numel	矩阵或者单元数组中元素的个数	ones	产生全是 1 的数组
rand	均匀分布随机数和数组	randn	正态分布随机数和数组
zeros	建立一个全零矩阵	: (colon)	等间隔向量
特殊变量和常数			
ans	最近的答案变量	computer	确定 MATLAB 运行的计算机
eps	浮点相对精度	Inf	无穷大
i	虚数单位	inputname	输入参量名
NaN	不是一个数字	nargin	输入参量的个数
nargout	输出参量的数目	pi	圆周率
nargoutchk	有效的输出参量个数	realmax	最大正浮点数
realmin	最小正浮点数	varargin	实际传入的参量
varargout	实际返回的参量		
时间和日期			
calendar	日历	clock	表示为时间向量的当前时间
cputime	逝去的 CPU 时间	date	的日期字符串
datenum	连续日期数字	datestr	数据字符格式



续表

时间和日期			
datevec	数据组成部分	etime	流逝的时间
eomday	月底	now	当前日期和时间
tic, toc	开始, 停止定时器	weekday	星期几
矩阵操作			
cat	连接数组	diag	对角矩阵和矩阵对角线
fliplr	从左至右翻转矩阵	flipud	从上到下反转矩阵
repmat	复制一个数组	reshape	改造数组
rot90	将矩阵翻转 90 度	tril	矩阵的下三角
triu	矩阵的上三角	: (colon)	数组下标, 重组矩阵
向量函数			
cross	向量叉积	dot	向量点积
intersect	两个的向量的交集	ismember	检测一个集合的元素
setdiff	返回两个向量的差集	setxor	两个向量的异或集
union	两个向量的并集	unique	一个向量的所有单一元素

表 A-9

坐标变换函数

坐标变换函数	
cart2pol	笛卡儿坐标转换为极坐标或者圆柱坐标
cart2sph	将笛卡儿坐标转化为球坐标
pol2cart	将极坐标或者圆柱坐标转换为笛卡儿坐标
sph2cart	将球坐标转换为笛卡儿坐标

表 A-10

特殊数学函数

特殊数学函数			
airy	Airy 函数	besselh	第三类 besselh 函数 (Hankel 函数)
besseli	修改后的 Bessel 函数	besselj	Bessel 函数
beta, etainc	Beta 函数	ellipj	雅克比椭圆函数
ellipke	第一类和第二类完全椭圆积分	erf, erfc	误差函数
expint	指数积分函数	factorial	阶乘函数 (n!)
gamma	伽马函数	legendre	关联 Legendre 函数
pow2	2 为底的指数函数	rat, rats	有理分数近似

表 A-11

矩阵函数——数值线性代数

矩阵分析			
cond	计算矩阵条件数	condeig	特征值表示的条件数
det	矩阵行列式值	norm	向量或者矩阵的范数
null	空矩阵	orth	一个矩阵的扩展空间
rank	矩阵的秩	rcond	矩阵逆条件数估计
rref,rrefmovie	精简行格式矩阵	subspace	两个子空间的角度
trace	对角线元素的和——迹		
线性方程			
chol	Cholesky 分解	inv	矩阵求逆
lsconv	协方差已知情况下最小平方解	lu	上下三角分解 (LU)
lsqnonneg	非负最小平方	minres	最小残余方法
pinv	矩阵伪逆	qr	正交三角分解 (QR)
symmlq	对称下三角方法		
特征值和奇异值			
banlance	提高特征值计算的精度	cdf2rdf	复数对角形式转换为实数块对角形式
eig	特征值和特征向量	gsvd	生成奇异值分解
hess	矩阵的 Hessenberg 形式	poly	求设定根的特征多项式
qz	生成特征值的 QZ 分解	rsf2csf	变实分块对角阵为复对角形式
schur	实 Schur 形式转换为复数 Schur 形式	svd	奇异值分解
矩阵函数			
expm	矩阵对数	funm	一般矩阵的计算
logm	矩阵对数	sqrtm	矩阵均方根
低层函数			
qrdelete	从 QR 分解删除列	qrinsert	在 QR 分解插入列

表 A-12

数据分析和傅立叶变换

基本操作			
cumprod	累积	cumsum	累加
cumtrapz	累计梯形法计算数值微分	factor	质因子
inpolygon	删除多边形区域内的点	max	一个数组最大元素
mean	数组的均值	median	数组的中值
min	一个数组的最小元素	perms	所有可能的置换
polyarea	多边形区域	primes	生成质数列表

续表			
基本操作			
prod	数组元素的乘积	rectint	矩形交集区域
sort	按升序排列矩阵元素	sortrows	按升序排列行
std	标准偏差	sum	数组元素的和
trapz	梯形数值积分	var	方差
有限差值			
del2	离散拉普拉斯	diff	插值和微分估计
gradient	数值梯度		
相关			
corrcoef	系数的相关	cov	协方差矩阵
滤波和卷积			
conv	卷积和多项式乘法	conv2	二维卷积
deconv	反卷积和多项式除法	filter	IIR 或者 FIR 滤波器
filter2	二维数字滤波器		
傅立叶变换			
cplxpair	将复数值分类为共轭对	fft	一维的快速傅立叶变换
fft2	二维快速傅立叶变换	fftshift	将 FFT 的 DC 分量移到频谱中心
ifft	一维快速傅立叶反变换	ifft2	二维傅立叶反变换
ifftn	多维快速傅立叶反变换	ifftshift	反 FFT 偏移
nextpow2	最靠近的 2 的幂次	unwrap	校正相位角

表 A-13 多项式和插值

多项式			
conv	卷积和多项式乘法	roots	多项式的根
poly	具有设定根的多项式	polyder	多项式微分
polyeig	多项式特征值问题	polyfit	多项式曲面拟合
polyint	解析多项式积分	polyval	多项式求值
polyvalm	矩阵变量多项式求值	residue	部分分式展开（留数计算）
插值			
convhull	凸包	convhulln	多维凸包
delaunay	Delaunay 三角	delaunay3	三维 delaunay 网格
delaunayn	多维 delaunay 网格	dsearch	寻找最近点
dsearchn	多维最近点搜索	griddata	数据网格
griddata3	数据网格和超平面拟合	griddatan	数据网格和超平面拟合（维数大于 2）

续表

插值			
interp1	一维数据插值	interp2	二维数据插值
interp3	三维数据插值	interpft	使用 FFT 的一维插值
interpn	多维数据插值	meshgrid	为 3 维点生成 X 和 Y 网格
ndgrid	生成多维函数和插值的数组	pchip	分段三次 Hermite 插值多项式
ppval	分段多项式估值	spline	三次样条数据插值
tsearch	寻找封闭的 Delaunay 三角	tsearchn	多维最近单一搜索
voronoi	Voronoi 图表	voronoin	多维 Voronoi 图表

表 A-14

泛函——非线性数值方法

非线性数值方法			
bvp4c	为常微分方程求解两点边界问题	bvpget	从 BVP 选项结构提取参数
bvpinit	形成 bvp4c 的初始猜测	bvpset	建立或者更替 BVP 选项结构
bvpval	估算 bvp4c 计算出的解	fminbnd	最小化单一变量函数
dblquad	double 积分的数值估算	fminsearch	最小化几个变量的函数
fzero	寻找单一变量的零点	odeget	从 ode 选项结构提取参数
odeset	建立和更替 ODE 选项结构	optimget	获取优化选项结构参数值
optimset	建立或者编辑选项参数结构	pdepe	解决初始边界值问题
pdeval	估算 pdepe 计算出的解	quad	低阶法计算数值微分
quadl	Lobatto 法求数值积分	vectorize	向量化表达

表 A-15

稀疏矩阵函数

基本稀疏矩阵			
spdiags	抽取和建立稀疏对角矩阵	speye	稀疏单位矩阵
sprand	稀疏均匀分布随机数矩阵	sprandn	稀疏正态分布随机数矩阵
sprandsym	稀疏对称随机矩阵		
完全矩阵和系数矩阵之间变换			
find	找出矩阵的非零元素的值及其下标	full	将稀疏矩阵转换为完全矩阵
sparse	建立稀疏矩阵	spconvert	从稀疏矩阵外部格式输入矩阵
处理稀疏矩阵的非零元素			
nnz	矩阵非零元素的个数	nonzeros	矩阵的非零元素
nzmax	分配给矩阵非零元素的存储空间	spalloc	为稀疏矩阵分配空间
spfun	将函数应用到矩阵的非零元素	spones	用 1 替代非零矩阵元素
可视化稀疏矩阵			
spy	可视化稀疏模式		

续表

排序算法			
colamd	列近似最小程度置换	colmmd	最小程度置换稀疏列
colperm	基于非零数目的列置换	dmperm	Dulmage-Mendelsohn 分解
randperm	随机置换	symamd	对称近似最小程度置换
symmmd	稀疏对称最小程度排列		
范数、条件数和秩			
condest	1 范数矩阵条件数估计	normest	2 范数估计
线性方程的稀疏系统			
bicg	双共轭梯度方法	bicgstab	双共轭稳态方法
cgs	共轭梯度平方法	cholinc	稀疏不完全 Cholesky 划分
cholupdate	秩为 1 更新到 Cholesky 分解	gmres	生成最小残余方法
lsqr	范数方程的共轭梯度实现	luinc	不完全 LU 矩阵分解
pcg	预处理剃度共轭方法	qmr	Quasi-Minimal 残余方法
qr	正交三角分解	qrupdate	将秩 1 更新为 QR 分解

表 A-16

音频处理函数

一般音频函数			
lin2mu	将线性音频信号转换为 mu-law	mu2lin	mu-law 转换为线性音频信号
sound	将向量转化为声音	soundsc	度量数据并作为声音播放
.wave 音频函数			
wavplay	在 PC 话音输出器件播放记录的音频	wavread	读 .wave 音频文件
wavrecord	使用 PC 话音输入设备记录音频	wavwrite	写 .wave 音频文件

表 A-17

字符串函数

通用函数			
abs	绝对值和复数模长	eval	解释包含 MATLAB 字符串的字符串
real	复数值的实数部分	strings	MATLAB 字符串句柄
字符串到函数句柄转换			
func2str	从一个函数句柄建立函数名字符串	str2func	从函数名建立一个函数句柄
字符串处理			
deblank	去掉字符串末尾的空格	lower	将字符串变成小写
findstr	在另一个字符串里查找一个字符串	strcat	字符串连接
strcmp	字符串比较	strjust	判断一个字符串组
strmatch	寻找一个字符串的可能匹配	strncmp	比较字符串的前 n 个字符
strncmppi	比较字符串前 n 个字符, 忽略大小写	strrep	字符串搜索和替换

续表

字符串处理			
strtok	字符串的第一个标记	strvcat	字符串的竖直连接
symvar	决定一个表达式的符号变量	texlabel	从一个字符串产生 TeX 格式
upper	将字符串转化为大写形式		
字符串到数字的转换			
char	建立字符数组	int2str	整数到字符串转换
mat2str	将矩阵转化为字符串	num2str	数字到字符串的转换
sprintf	写格式化过的数据到字符串	sscanf	在格式控制下读字符串
str2double	将字符串转化为 double 精度数值	str2mat	字符串到矩阵的转换
str2num	字符串到数字的转换	bin2dec	二进制数到十进制的转换
dec2bin	十进制数到二进制数转换	hex2dec	十六进制到二进制的转换
dec2hex	十进制到 16 进制的转换	hex2num	十六进制到 double 类型数的转换

表 A-18

文件输入/输出函数

文件打开和关闭			
fclose	关闭一个或者多个文件	fopen	打开文件或者获得打开文件的信息
无格式化输入、输出			
fread	从文件读二进制数据	fwrite	向文件写二进制数据
格式化输入和输出			
fgetl	取下一行作为字符串，无终止符	fgets	取下一行作为有终止符的字符串
fprintf	写格式化数据到文件	fscanf	从文件读格式化数据
文件定位			
feof	检测是否到了文件结尾	ferror	向 MATLAB 查询输入和输出错误
frewind	重绕一个打开的文件	fseek	设置文件定位器的位置
ftell	获得文件定位器的位置		
特殊文件输入/输出			
dlmread	将 ASCII 无限制文件读入矩阵	dlmwrite	写一个矩阵到 ASCII 无限制文件中
hdf	HDF 接口	imfinfo	返回一个图形文件的信息
imread	从图形文件读图像	imwrite	写数据到图形文件
strread	从一个字符串读格式化数据	textread	从文本文件读格式化数据

表 A-19

位逻辑运算函数

位逻辑运算函数			
bitand	按位求与	bitcmp	对位求补
bitor	按位求或	bitmax	最大浮点整数

续表

位逻辑运算函数			
bitset	设置 bit	bitshift	比特移位
bitget	获得位	bitxor	按位异或

**表 A-20** 单元数组函数

单元数组函数			
cell	建立单元数组	cellfun	将函数应用到单元数组的每个元素
cellstr	从字符数组建立字符串单元数组	cell2struct	单元数组到结构数组的变换
celldisp	显示单元数组内容	cellplot	图形显示单元数组的结构
num2cell	转换数值数组到一个单元数组		

**表 A-21** 多维数组函数

多维数组函数			
cat	连接数组	flipdim	按一个指定的维翻转矩阵
ind2sub	来自线性索引的下标	ipermute	多维数组的维数反置换
ndgrid	为多维函数或者插值生成数组	ndims	数组的维数
permute	重排多维数组的维数	reshape	重构数组形式
shiftdim	偏移维数	squeeze	去除单一维

**表 A-22** 绘图和数据可视化函数

基本绘图和图形			
bar	竖直条图表	barh	水平条图表
hist	绘制直方图	histc	直方图计数
hold	保持当前图形	loglog	绘制图形使用 log-log 坐标
pie	饼状图	plot	绘制向量或矩阵
polar	极坐标绘图	semilogy	半对数绘图 (y 轴对数坐标)
semilogx	半对数绘图 (x 轴对数坐标) 对数	subplot	绘制子图
三维绘图			
bar3	数值 3D 竖条图	bar3h	水平 3D 条状图
comet3	3D 彗星图	cylinder	生成圆柱体
fill3	绘制填充的 3D 多边形	plot3	在 3 维空间绘制线和点
quiver3	3D 震动 (速度) 图	slice	体积薄片图绘制
sphere	生成球	stem3	绘制离散表面数据
waterfall	绘制瀑布	trisurf	三角表面

续表

绘制注释栅格			
clabel	增加轮廓标签到等高线图中	datetick	数据格式标记
grid	在 2D 和 3D 图上添加方格线	gtext	使用鼠标将文本放置在 2D 图上
legend	线或点的图注	plotyy	绘制 Y 轴标记在左边和右边的图形
title	添加 2D 和 3D 图的标题	xlabel	X 轴的标签
ylabel	Y 轴标签	zlabel	3D 图的 Z 轴标签
表面、网格和等高线			
contour	画等高线图	contourc	等高线计算
contourf	填充的等高线绘图	hidden	设置网格消影方式
meshc	连接网格/等高线	mesh	具有参考轴的 3D 网格
peaks	具有两个变量的采样函数	surf	3D 阴影表面图
surface	建立表面低层对象	surfc	海浪和等高线的结合
surfl	具有光照的 3D 阴影表面	trimesh	三角网格图
数量可视化			
coneplot	在 3D 向量场将速度向量绘成锥体	contourslice	在切片几何平面画等高线
curl	计算一个向量场的曲率和角速度	divergence	计算一个向量场的散度
flow	生成标量数量数据	isocolors	计算同平面顶点的颜色
isonormals	计算同平面顶点的范数	isosurface	从量的数据抽取同平面数据
reducepatch	减少补片的面数	reducevolume	减少数量数据集的元素个数
shrinkfaces	减少补片面的大小	smooth3	光滑 3D 数据
stream2	计算 2D 流线数据	stream3	计算 3D 流线数据
streamline	从 2D 或者 3D 向量数据绘制流线	streamparticles	从向量数据画流体
treamtube	从向量数据绘制流管		
查看控制			
camdolly	移动照相机位置和指向	camlookat	查看设定的对象
camorbit	围绕照相机指向旋转	campan	围绕照相机位置转动指向
campos	设定或者获得照相机位置	camroll	围绕视轴旋转照相机
camproj	决定投影类型	camtarget	设置或者得到照相机指向
camup	设置或者获得照相机向上向量	camva	设置或者得到照相机视角
camzoom	将照相机放大或者缩小	pbaspect	设置或者得到绘图盒外形比率
daspect	设置或者得到数据形状比率	view	3D 图形视点说明
viewmtx	生成视图变换矩阵	xlim	设置或者获得当前 X-轴的限度
ylim	设置或者获得当前 y 轴的限度	zlim	设置或者获得 z 轴的限度
光照			
camlight	生成或者定位光线	light	光线对象建立函数
lighting	光照模式	lightangle	在球坐标定位光照



续表			
透明性			
alpha	设置或查询当前轴的对象透明性	alphamap	设定图像的透明性
颜色操作			
brighten	加亮或者变暗色彩匹配	caxis	伪色轴制式
colordef	建立色彩的缺省	colormap	设置色彩对照表
graymon	灰度图形的缺省设置	hsv2rgb	HSV 到 RGB 的转换
rgb2hsv	RGB 转换为 HSV	rgbplot	绘制彩色匹配
shading	彩色阴影模式	spinmap	选择色彩匹配
surfnorm	3D 表面法线	whitebg	改变绘图的轴背景色彩
色彩匹配			
autumn	红黄色彩匹配的阴影	bone	具有淡蓝色的灰度制式色彩匹配
contrast	提高图像对比度的灰度色彩匹配	cool	青色和洋红色色彩匹配的阴影
copper	线性紫铜色调匹配	flag	红、白、蓝和黑交替的色彩匹配
gray	线性灰度色彩匹配	hot	黑-红-黄-白色色彩匹配
hsv	HSV 色彩匹配	jet	HSV 的变形
lines	线性色彩匹配	prism	棱镜色彩匹配
spring	洋红和黄的色彩匹配的阴影	summer	绿和黄的色彩匹配的阴影

## 参 考 文 献

- [1] 王可定, 计算机模拟及其应用. 南京: 东南大学出版社, 1997
- [2] 顾启泰, 离散事件系统建模与仿真. 北京: 清华大学出版社, 1999
- [3] MATLAB Help document. The Mathworks. Inc, 1999
- [4] 李人厚等译校, 精通 MATLAB 综合辅导与指南. 西安: 西安交通大学出版社, 1997
- [5] 王立宁等, MATLAB 与通信仿真. 北京: 人民邮电出版社, 1999

